MASTER EN SCIENCES INFORMATIQUES

Un système d'aide à la simulation à évènements discrets en Fortran

Nisen, Jean-Mathieu; Roulin, José

Award date:
1981

Awarding institution:
Universite de Namur

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Institut d'Informatique

ANNÉE ACADÉMIQUE 1980-1981

UN SYSTÈME D'AIDE À LA SIMULATION À ÉVÉNEMENTS DISCRETS EN FORTRAN

1e partie

Jean-Mathieu NISEN
et
José ROULIN

Mémoire présenté
en vue de l'obtention du grade de
Licencié et Maître
en Informatique

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Institut d'Informatique

ANNÉE ACADÉMIQUE 1980-1981

UN SYSTÈME D'AIDE À LA SIMULATION À ÉVÉNEMENTS DISCRETS EN FORTRAN

1e partie

Jean-Mathieu NISEN
et
José ROULIN

Mémoire présenté
en vue de l'obtention du grade de
Licencié et Maître
en Informatique

TABLE DES MATIERES.

PREMIERE PARTIE.

DEUXIEME PARTIE.

TABLE DES FIGURES.

INTRODUCTION

ET

REMERCIEMENTS

INTRODUCTION ET REMERCIEMENTS.

Le document de base précisant le sujet de notre travail est ainsi conçu: il est demandé de "créer une bibliothèque de sous-programmes accessibles à partir d'un programme FORTRAN et reprenant les primitives essentielles du langage SIMULA. Ce sujet s'adresse à deux étudiants. La bibliothèque de sous-programmes devrait fonctionner sur le DEC 20".

*

1.- Disons d'abord que nous avons cru bon d'appeler SIMUFOR la bibliothèque que nous tentions de créer; est-il nécessaire d'insister sur le fait que le nom de SIMUFOR évoque tant la simulation que le langage FORTRAN ?

Nous nous sommes donc attachés à résoudre le problème qui nous était posé et, pour ce faire, avons implémenté, en FORTRAN, les éléments de base orientés simulation de SIMULA. Ceci nous a conduits à utiliser aussi deux langages d'assemblage: le COMPASS sur CDC puis l'ASSEMBLEUR sur le DEC. Dépassant ensuite quelque peu le sujet proposé, nous avons, en outre, suivi la voie du Prof. VAUCHER lorsqu'il a créé GPSSS et introduit, dans notre travail, les primitives essentielles de GPSS, afin d'accroître les possibilités de SIMUFOR.

C'est en totale collaboration que nous avons réalisé toute l'analyse fonctionnelle et l'implémentation, c'est-à-dire la partie "recherche" de SIMUFOR; de même, nous avons également conçu ensemble l'introduction et la conclusion (chap. 5).

Il s'est cependant avéré nécessaire de nous répartir les tâches de la rédaction, pratiquement impossibles à réaliser à deux:

- Jean-Mathieu NISEN a rédigé la partie bibliographique (chap. 1 et 2): on pourrait croire que nous avons donné à celle-ci une importance relative trop grande; néanmoins, il est apparu qu'elle constituait,

comme toute étude bibliographique d'ailleurs, la base même d'un travail personnel.

- D'un autre côté, José ROULIN a mis en forme la description des principes de base de l'implémentation (chap. 3).

- Nous nous sommes répartis la rédaction du chap. 4 (description, vue sous l'angle fonctionnel et sous l'aspect implémentation, des divers sous-programmes) de la façon suivante:

  * J. ROULIN s'est attaché à la description des sous-programmes d'ordre général, de traitement de listes, de "storages", de "régions", de statistiques ainsi que les sous-programmes Assembleurs;

  * De son côté, J-M. NISEN s'est vu confier la description des sous-programmes de gestion des processus, de "facilités", de "groupes", d'histogrammes, de nombres aléatoires, de "debugging", de traçage et d'erreur.

- Le côté rédactionnel de l'introduction et du chap. 5 (conclusions) a, enfin, été assuré par J-M NISEN.

N.B. Aux 5 chapitres précités, est jointe une ANNEXE qui regroupe l'ensemble des sous-programmes Fortran et Assembleur ainsi que quelques exécutions.

Nous avons constitué les fondements du travail présenté ci-après au cours d'un séjour de cinq mois au Département de Recherche Opérationnelle et d'Informatique de l'Université de Montréal grâce aux conseils éclairés du Prof. J.G. VAUCHER. Ce dernier était particulièrement bien placé pour diriger notre travail étant donné la grande expérience qu'il possède dans le domaine de la Simulation. Notre SIMUFOR s'inspire en effet des langages SIMULA et GPSS. Or le Prof. VAUCHER avait déjà réalisé une extension à SIMULA (appelée GPSSS), permettant une manipulation aisée des concepts de GPSS en SIMULA, et a de plus étendu la capacité de PASCAL en lui associant un certain nombre de sous-programmes facilitant la rédaction de programmes de simulation. Il était donc très au courant des pièges auxquels le sujet de notre mémoire nous exposait. Remarquons dès maintenant que le FORTRAN, plus ancien que le PASCAL, ne dispose pas des outils qui en font l'attrait et pose ainsi à ses utilisateurs des problèmes qu'il serait vain de sous-estimer.

2.- Au terme de ce travail, nous souhaitons remercier tous ceux qui, à quel que niveau que ce soit, ont contribué à sa réalisation.

Nous voulons citer tout d'abord le Directeur, les Professeurs et les Membres du Personnel Scientifique de l'Institut d'Informatique des Facultés Universitaires Notre Dame de la Paix qui nous ont formés dans le domaine combien spécialisé et en évolution constante de l'Informatique. En outre, c'est à sa juste valeur que nous apprécions l'autorisation qui nous a été donnée de pouvoir disposer de l'ordinateur des Facultés dans la mesure de nos besoins.

Nous tenons encore à dire à M. le Prof. J. FICHEFET combien nous avons apprécié les nombreux conseils qu'il nous a prodigués tout au long de ce travail et en particulier lors de sa rédaction. Nous remercions aussi Mme M. NOIRHOMME et M. P. LAMBION pour leurs encouragements et suggestions.

D'un autre côté, il nous est bien agréable de dire que nous avons trouvé beaucoup de compréhension et d'aide auprès des membres du personnel du Centre de Calcul des Facultés de Namur: nous les en remercions vivement.

Notre sincère reconnaissance va à M. le Prof. VAUCHER de l'Université de Montréal qui a accepté que nous nous intégrions à la vie de son service et que nous lui demandions les conseils dont nous avions besoin, et ce, malgré le peu de temps dont il disposait. Il nous a guidés pendant toute la période que nous avons passée à Montréal et ne nous a jamais ménagé son aide et ses conseils.

Enfin, nous voudrions encore exprimer ici toute notre gratitude à Melle L. ROY et à ses collaboratrices de l'Université de Montréal: C'est grâce à leur amabilité qu'ont pu être résolus les problèmes administratifs que pose un séjour d'assez longue durée dans un pays étranger, fût-il ami.

CHAPITRE 1

QU'EST-CE QUE LA SIMULATION ?

1. QU'EST CE QUE LA SIMULATION?

1.1. INTRODUCTION.

Dans ce chapitre, nous croyons opportun de rassembler un certain nombre de données qui peuvent paraître élémentaires à tout qui connaît un peu la simulation et même à tout informaticien. On pourrait donc considérer que ces généralités ne méritent pas de figurer dans un travail de fin d'études de licence en informatique. Cependant, en présentant d'abord un ensemble de définitions simples, nous espérons éviter toute interprétation erronée de mots auxquels le langage technique attribue une signification qui s'écarte parfois sensiblement de celle reconnue par l'usage.

En conséquence, dans les pages qui vont suivre et à titre d'introduction, nous suivons successivement les étapes ci-après:

1) nous définissons d'abord le processus théorique de la simulation, travail plus complexe qu'il n'y paraît à première vue;

2) nous nous attardons quelque peu sur les deux éléments de base de la simulation: le système et le modèle;

3) nous recherchons les relations qui existent entre ces éléments constitutifs de la simulation, ce qui en constitue au fond une approche primaire;

4) nous distinguons ce processus d'autres cheminements de la pensée (observation directe et analyse mathématique notamment) et tentons de préciser les limites de chacun d'eux;

5) nous schématisons les types de simulation possibles ce qui nous conduit d'ailleurs à préciser de façon nette le domaine d'application de notre travail;

6) nous faisons la synthèse des avantages et des inconvénients que nous voyons à la simulation dans son sens le plus large et dans celui qui fait l'objet de notre travail en particulier;

7) enfin, nous décrivons rapidement les principales étapes de la simulation.

1.2. DEFINITIONS.

1.2.1. Simulation.

Le mot "simuler" vient [11] du verbe latin simulare, c'est-à-dire copier, reproduire, imiter. La notion de "copier" n'est pas si peu adéquate qu'on pourrait le croire à première vue car, par la simulation (J. SMITH [16]), nous tentons de copier le comportement d'un système, dont l'approche mathématique est malaisée, au moyen d'un autre système (modèle) qu'il sera possible de traiter numériquement. On évoque donc, à ce niveau déjà, la notion d'analogie proposée par J. BUREAU [3].

Nous pensons que cette définition complète bien celle également proposée par J. BUREAU [3]:

> " la simulation est la représentation du comportement d'un ensemble par un autre ensemble de nature mathématique ou physique".

De leur côté, J.G VAUCHER et P. BRATLEY [18] proposent d'une manière assez semblable:

> " la simulation est une technique qui utilise l'expérimentation sur des modèles pour étudier le comportement de systèmes complexes ".

Dans ces deux références, l'accent est donc mis sur le caractère dynamique de la simulation puisque c'est le comportement d'un ensemble qu'on représente et non uniquement cet ensemble lui-même. C'est, comme nous l'évoquerons plus tard, un des avantages de la simulation: celui d'étudier les effets de changements de divers types sur le fonctionnement d'un système. Nous voyons d'autre part que les systèmes cités par J.G VAUCHER et al. [18] correspondent au premier ensemble mentionné par J. BUREAU [3], et les modèles au second ensemble.

Une définition mathématique (C.W. CHURCHMAN in [14]) cerne bien le problème mais ne l'explicite pas:

" 'x simule y' est vrai
si et seulement si
a) x et y sont des systèmes formels
b) y représente le système réel
c) x représente une approximation du système réel
d) les lois de validité dans x ne peuvent être entachées d'erreur ".

La simulation apparaît donc bien malaisée à définir de façon pratique car les auteurs qui en parlent introduisent des concepts divers sur la signification desquels ils ne sont pas toujours d'accord.

D'autre part si l'on découpe de façon simple [1] les techniques scientifiques en quatre groupes et ce en fonction de la nature des problèmes et du genre de modèles utilisés pour les résoudre, on en arrive à la classification de la figure 1-1 ci-après.

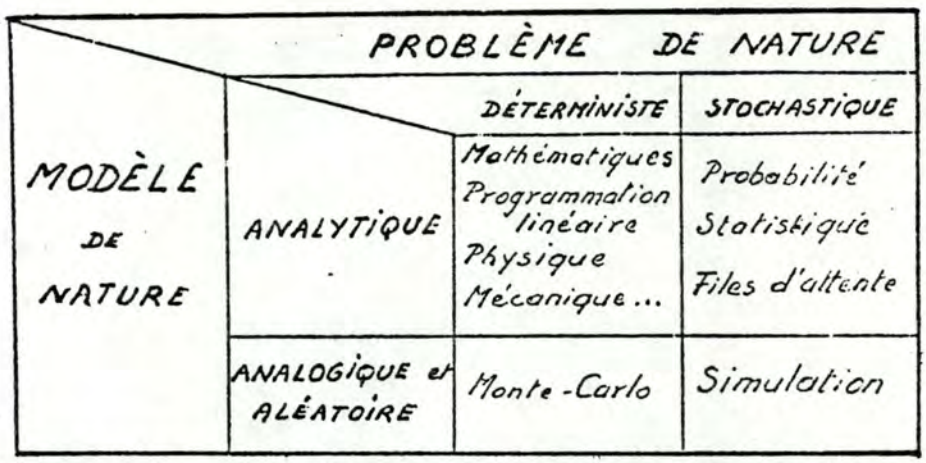


Figure 1-1: Classification des techniques scientifiques [1].

La classification ne nous satisfait pas davantage que les définitions ci-dessus présentées. En effet, il semble au vu de cette classification (trop simple) que, d'une part, la simulation ne puisse s'appliquer à des problèmes de nature déterministe et que, d'autre part, "Monte-Carlo" ne soit pas un type de simulation, affirmation que n'admettent ni FISHMAN [7] ni Th. NAYLOR et al. [14]; ils définissent justement la méthode Monte-Carlo comme une technique de simulation applicable à des problèmes qui ont une base stochastique ou probabiliste. Enfin, J. AGARD et al. [1] se demandent s'il ne vaut pas mieux, pour résoudre certains problèmes dont la représentation mathématique est possible mais complexe de préférer la simulation à l'étude analytique! Dans ce cas d'ailleurs, ils rencontreraient l'optique de Th. NAYLOR et al. [14] qui

voient comme un des avantages de la simulation la possibilité de vérifier les solutions analytiques.

Pour ces raisons, J. AGARD et al. [1] préfèrent la définition suivante:

> "une simulation reproduit à vitesse accélérée l'évolution temporelle d'un système en tenant généralement compte d'aléas pour donner, sur le comportement du système, des renseignements que d'autres méthodes ne fourniraient pas si ce n'est à un prix de revient supérieur ".

Nous voyons que J. AGARD et al. [1] introduisent notamment la notion de reproduction accélérée d'un phénomène temporel, mais nous nous demandons alors avec J. BUREAU [3] s'il n'est pas préférable de dire:

> " les systèmes de simulation sont des modèles dynamiques... qui s'écartent de la réponse en temps réel par dilatation ou mieux par contraction de l'échelle des temps."

Pour terminer cette tentative très incomplète de définition générale du concept "simulation", notons que M.S. SHUBIK, cité par Th. NAYLOR et al. [14], y ajoute la notion d'impossibilité:

> "... le modèle est soumis (grâce à la simulation) à des manipulations qui seraient impossibles, trop coûteuses ou irréalisables..." (par une autre méthode)

et que J. SMITH [16] propose:

> "Comme alternative à l'analyse mathémathique, nous pouvons nous tourner vers des méthodes numériques (simulation); nous pouvons procéder comme suit: nous supposons quelque état initial (condition) pour le système étudié, et nous utilisons des lois de changement (règles) en vue d'évaluer les états (positions) à travers lesquels le système progresse pendant une période de temps donnée".

EN RESUME, nous pouvons dire avec J. FICHEFET [6] que, étant donné les points de vue différents auxquels se placent les divers auteurs, il y a peut-être autant de définitions de la simulation qu'il y a de livres ou d'articles traitant le sujet et même qu'il y a de disciplines recourant à la simulation; deux traits communs à toutes les définitions apparaissent:

- "une simulation est une reconstitution artificielle d'un phénomène réel, et

- cette reconstitution doit être suffisamment fidèle pour que l'on puisse considérer les renseignements qu'elle fournit comme étant valables pour le phénomène réel" [6].

N.B: A partir de ces définitions que nous avons tenté de présenter d'une façon très générale, il est naturellement possible de proposer pour le terme simulation toute une série de sens restrictifs adaptés à des cas concrets bien précis. Il est vraisemblable d'ailleurs que c'est parce que chacun des auteurs précités poursuit un but bien particulier que la définition qu'il donne de la simulation (ou l'idée qu'il s'en fait) est limitée aux éléments qu'ils proposent. Passer en revue tous les sens restrictifs concrets sortirait du cadre du présent travail et la liste que nous pourrions en proposer resterait toujours loin d'être complète. Nous nous bornerons donc, dans notre travail, à l'aspect de la SIMULATION qui concerne plus particulièrement celle DES MODELES PROGRAMMES SUR ORDINATEURS DIGITAUX.

*

En première approximation, il apparaît donc qu'une simulation s'effectue en trois étapes:

1. détermination du système
2. création d'un modèle de simulation
3. recherche des relations entre systèmes et modèles.

Ces trois étapes seront envisagées successivement et rapidement au cours des paragraphes 1.2.2, 1.2.3 et 1.3 ci-après.

1.2.2. Système.

Il nous paraît que la terminologie proposée par J.G VAUCHER et P. BRATLEY [18] dans leur ouvrage très documenté est la mieux adaptée au problème que nous devons traiter. Nous y faisons une référence particulière dans les paragraphes suivants.

1) Le système est un ensemble de parties cohérentes mais comme l'observe J.G. VAUCHER,

> "il reste un artifice d'abstraction: les choses ne sont pas intrinsèquement réparties en systèmes. C'est une partie de la réalité qu'on choisit, dans un but quelconque de considérer comme un tout... C'est une collection d'objets (entités ou composantes) ayant certaines caractéristiques (attributs) qui prennent part à des activités".

M.G. HARTLEY [10] introduit la notion d'environnement (citée aussi par J.G VAUCHER et al.) en disant qu'il faut non seulement analyser la conception des différents éléments qui forment l'ensemble d'un système, mais aussi considérer ceux qui apparaissent en relation avec tous les autres; que le système proposé doit donc être étudié à un niveau plus élevé, en relation avec son environnement; en d'autres termes qu'il est nécessaire d'inclure au système les facteurs économiques, sociologiques, de l'environment et ceux de "l'ingéniérie."

2) L'état d'un système est une description complète de la valeur de toutes les caractéristiques (attributs) du système et des actions à un moment donné. Un système dynamique (voir définition plus loin) s'étudie souvent par l'examen de son état à des intervalles de temps successifs, ce qui permet l'utilisation d'ordinateurs digitaux pour la simulation.

3) Diverses classifications des systèmes sont proposées par les auteurs. Nous pensons pouvoir les résumer comme suit en les rangeant selon les points de vue envisagés:

1. selon le temps:

   - si ses attributs (caractéristiques) varient dans le temps, le système est appelé dynamique. Sinon, on parle d'un système statique;
   - si dans le cas d'un système dynamique, après l'initialisation, les attributs continuent à varier de manière continue dans le temps, le système est dit continu; sinon, il est dit discret;

     NB: un changement à l'état du système est appelé "événement".

2. selon la présence de facteurs aléatoires:

   si ceux-ci existent, le système est dit stochastique; sinon, on parle d'un système déterministe;

3. selon l'environnement:

si le système n'est pas soumis à l'environnement, il est dit fermé; sinon, il est considéré comme ouvert.

REMARQUES:

1. Les systèmes simulés sont le plus souvent discrets (et par conséquent dynamiques) et stochastiques.

2. Comme nous le verrons plus loin, à chaque système correspond un langage de programmation; ainsi aux systèmes discrets s'appliquent des langages tels que SIMULA, GPSS, SIMSCRIPT.

3. Les motivations de l'étude des systèmes sont diverses:

   - souci d'optimisation pour les systèmes artificiels
   - souci de compréhension pour les systèmes naturels
   - souci d'optimisation et de compréhension pour les systèmes sociaux, juridiques ou économiques.

1.2.3. Modèles

J.G VAUCHER et P. BRATLEY [18] définissent le modèle comme une "représentation d'un système". L'étymologie de "repraesentare" (reproduire) conduit à comprendre le terme de représentation comme l'image, le symbole, la personnification.

J. SMITH [16] estime que, en "copiant" un système de la vie réelle, nous

"obtenons, dans l'ordinateur, un modèle du système réel qui se comporte beaucoup comme celui-ci. Par "modèle", dans ce contexte, nous nous référons non pas à une entité qui ressemble physiquement au système de la vie réelle, mais à un autre qui le simplifie en un ensemble de variables qui représentent les faits principaux du système réel et en un ensemble d'instructions représentant les lois (règles de décisions) qui déterminent comment ces faits sont modifiés lorsque le temps progresse..."

ROSENBLUETH et WIENER in [14] voient dans le modèle une "abstraction d'un

système réel".

Le sens premier de "abstrahere" (tirer de) nous paraît donc un élément important: le modèle serait l'image d'un système mais il en serait également issu; il serait donc aussi semblable que possible au système mais devrait être plus simple: il ne peut pas tenir compte de tous ses éléments, se borne à ses aspects pertinents et reste donc une approximation. Cette approximation doit évidemment être aussi fidèle à la réalité que possible mais elle ne doit pas comporter des détails qui, en nombre excessif, risqueraient de cacher l'essentiel.

Les deux grandes caractéristiques d'un modèle sont donc réalité et simplicité (relatives): le modèle doit être une approximation suffisamment rigoureuse du système réel; il doit en contenir la plupart des aspects importants et il ne doit pas être à ce point complexe qu'il soit malaisé à comprendre ou à utiliser.

Terminons cependant en disant que:

> "c'est le système qui nous intéresse mais c'est le modèle que l'on étudie". [18]

1.3. RELATION SYSTEME MODELE

Nous venons de voir que, bien que ce soit le système qui nous intéresse, c'est le modèle que l'on étudie.

Mais alors se pose le problème: "comment étudier le modèle"?

Diverses voies sont possibles et applicables selon le cas:

1. l'observation directe
2. l'analyse mathematique
3. la simulation.

1.3.1. L'observation directe.

L'observation directe conduit dans bien des cas, à des coûts d'expérimentation aberrants: il faut constituer de toute pièce une réalisation pratique du problème qui se pose. Dans beaucoup de domaines, les crédits consacrés à la recherche excluent presque automatiquement cette technique. Souvent, en outre, aucune optimisation à partir des cas particuliers réalisés n'est possible: l'étude ne peut dépasser le stade de leur simple comparaison.

1.3.2. L'analyse mathématique.

Lorsqu'elle est possible, l'analyse mathématique apparaît comme la solution sinon la plus élégante du moins très souvent préférable à toute autre méthode, simulation comprise, ne serait-ce que pour son prix réduit [6]. Il faut cependant observer que, d'une part, le nombre de modèles pour lesquels il existe une solution mathématique est assez limité et que, d'autre part, l' introduction d'hypothèses simplificatrices, pour transformer un problème réel en un problème traitable réduit la valeur des solutions obtenues en fonction même du nombre et de l'importance des hypothèses restrictives dont il a dû être fait usage. (cf 1.5.1.6)

1.3.3. La simulation.

La simulation s'affirme donc comme une "technique de dernier recours", ce qui est loin de lui enlever de sa valeur, bien au contraire: son coût la réserve automatiquement à l'étude de problèmes pour lesquels aucune des deux autres méthodes ne s'avère satisfaisante. D'un autre côté, (et ceci n'est pas non plus susceptible d'en limiter son intérêt), la simulation peut même apparaître comme une étape préalable à l'élaboration d'un modèle analytique...!

On peut donc utilement se poser la question "Pourquoi faire appel à la simulation ?". La réponse apparaît immédiate si l'on considère avec J.G VAUCHER et P. BRATLEY [18] que la présence des caractéristiques suivantes dans un système entraîne souvent et presque ipso-facto l'utilisation de la simulation: non linéarité, présence de phénomènes transitoires ou de phénomènes aléatoires,rétroaction, ampleur. Ceci annonce déjà les principaux avantages de la simulation (cf 1.5)

De leur côté, J. AGARD et al. [1] observent que:

- le succès croissant de la simulation s'explique par le fait que celle-ci prend le relais d'autres techniques qui s'avèrent impuissantes à résoudre des problèmes d'une logique trop riche pour être ramenées à une formulation canonique simple;
- de plus, la simulation, en appliquant des règles logiques séquentielles, permet de reproduire pratiquement tout processus dont les règles d'évolution sont définies.

1.4. TYPES DE SIMULATION.

J. BUREAU [3] et M.G. HARTLEY [10] s'accordent à reconnaître deux types de simulation, la simulation analogique et la simulation digitale, du fait que ces deux simulations font appel aux ordinateurs portant le même nom. Parmi les simulations digitales, on distingue les simulations hardware et software dont la signification est évidente. En ce qui concerne ce dernier type (software), J.G VAUCHER et P. BRATLEY distinguent la simulation discrète (correspondant à des systèmes discrets) et la simulation continue (correspondant à des systèmes continus).

La simulation discrète étant étudiée au chap. 2, nous nous bornerons à esquisser une définition de la simulation continue: les activités prédominantes d'un système continu provoquent des modifications continues dans les caractéristiques (attributs) des objets (entités, composantes) du système; un simulateur continu est, selon J.G. VAUCHER,

> "simplement un programme pour la résolution numérique de systèmes d'équations différentielles".

LE SUJET DE NOTRE TRAVAIL nous conduit à n'utiliser que la simulation DIGITALE SOFTWARE A EVENEMENTS DISCRETS. Nous n'aborderons donc plus, dans les pages qui suivent, les simulations analogique ni digitale hardware, bien que les avantages et inconvénients de la simulation que nous présentons au par. 1.5 puissent s'appliquer aux divers types de celle-ci.

1.5. AVANTAGES ET INCONVENIENTS DE LA SIMULATION.

Il va de soi que si la simulation (prise dans un sens le plus général) rencontre le succcès qu'on lui connaît en ce moment, c'est qu'elle possède des avantages qui lui sont propres, c'est-à-dire que ne possède aucune autre méthode de résolution de systèmes complexes, utilisable à l'heure actuelle. Ces avantages en font un outil de travail si intéressant que l'on tente à l'heure actuelle de l'utiliser de façon pratique pour résoudre des problèmes étonnants par leur nombre, leur diversité et leur complexité.

A côté des avantages, les simulations présentent, cela va de soi, certaines limites (restrictions quant à leur domaine d'application) et peut-être même certains inconvénients.

Nous n'avons pas la prétention de donner ci-après une liste exhaustive de tous les avantages et inconvénients de la simulation d'autant plus que nous tenterons de rester sur le plan général et n' aborderons aucun problème pratique.

1.5.1. Avantages.

Nous avons déjà défini, au par. 1.3.3, les raisons qui conduisent à utiliser la simulation ainsi que ses principales possibilités. Plaçons-nous ici sous un angle plus descriptif et recherchons systématiquement tout ce qui peut être porté au crédit de la simulation. Observons en outre que les avantages cités ne s'excluent pas mutuellement mais peuvent se cumuler. Il va de soi, d'autre part que les avantages de la simulation s'identifient, au moins partiellement, à ceux de la modélisation. Un certain nombre de points repris ci-dessous apparaissent chez E. FISHMAN comme à porter à l'actif du modèle.

1) Rappelons que la raison principale qui conduit au choix de la simulation est son aptitude à la résolution de problèmes auxquels les autres méthodes sont incapables de donner une solution. Certains des autres avantages cités ci-après sont en relation directe ou indirecte avec ce fait.

2) La simulation permet d'aborder, en vue d'études et d'expérimentations, des problèmes caractérisés par des interactions complexes.

3) La simulation est, selon J. AGARD et al. [1], "aisée à écrire, à expliquer et à comprendre". Certains auteurs vont même jusqu'à dire qu'elle exige moins de connaissances théoriques que la résolution mathématique d'un problème.

4) Il est possible d'étudier les effets qu'apportent, sur le fonctionnement d'un système, des modifications au niveau de l'information, à celui de l'organisation et à celui de l'environnement; cette étude s'effectue en imposant les mêmes modifications au modèle (qui représente le système), en observant leurs conséquences puis en les transférant au système lui-même.
En d'autres termes, il est plus simple de "travailler" sur le modèle que sur le système et il est possible d'appliquer au second les résultats obtenus sur le premier.

5) Dans le même ordre d'idées, puisqu'elle permet de réitérer un grand nombre de fois l'évolution d'un système (à condition que l'on utilise un langage de programmation particulièrement bien adapté et que l'on choisisse convenablement les paramètres), la simulation permet d'apprécier la dispersion des résultats et de connaître les conséquences des modifications éventuellement apportées.

6) La création et l'exploration d'un modèle permet de mieux comprendre le système dont il est issu: en effet, selon J. AGARD et al. [1] le premier avantage de la simulation est "la facilité de compréhension qu'elle offre à l'utilisateur qui y verra les règles de la réalité clairement exprimées sans simplification et sans formulation mathématique complexe"..."La simulation sert ainsi à entraîner les responsables à mieux analyser leurs problèmes".

7) La création d'un modèle peut permettre de rechercher la nature et le comportement des variables majeures d'un système complexe.

8) La simulation est parfois utile pour scinder un système complexe en

sous-systèmes qui sont plus aisés à traiter par un analyste expert en la matière (HORGENTHALER cité par [14]).

9) La simulation permet d'étudier les systèmes dynamiques à vitesse réelle (temps réel), à vitesse accélérée, ou à vitesse réduite; nous avons, en effet, montré plus haut (1.2.1) qu'il peut être intéressant de réduire la vitesse du travail si cette réduction permet de mieux comprendre un phénomène qui se passe trop rapidement pour pouvoir être bien analysé.

10) La pratique de la simulation s'avère enfin un outil pédagogique important puisqu'elle conduit l'utilisateur à acquérir une habileté réelle au niveau de l'analyse des systèmes, des problèmes probabilistes et des décisions.

1.5.2. Inconvénients.

La simulation présente, à côté de nombreux avantages, un certain nombre d'inconvénients qui précisent dans une certaine mesure ses limites. Certes, ses inconvénients apparaissent sensiblement moins nombreux que ses avantages, mais leur importance n'est pas à négliger. Encore que J. AGARD et al. nous paraissent avoir pu dans un certain nombre de cas, réduire les conséquences des inconvénients de la simulation, notamment lorsqu'ils ont pu mettre en oeuvre des processus de régulation qui permettent d'atteindre un état stable à partir de conditions de départ mal étudiées. Ces auteurs parlent même de "simulations auto-adaptatives qui rectifient automatiquement les valeurs de certains paramètres en vue d'optimiser certains critères".

Les trois premiers inconvénients présentés sont inspirés de J. AGARD et al. [1] tandis que les suivants résument plutôt le point de vue de FISHMAN [7].

1) Le défaut le plus apparent semble être la longueur du temps nécessaire pour mettre au point et par conséquent les coûts de la technique de simulation. Contrairement aux solutions mathématiques de modèles analytiques, une fois les données du problème rassemblées, la solution n'est pas imminente, au contraire: les délais de programmation et de mise au point prennent parfois plusieurs mois.

2) Une conséquence de ce premier inconvénient est que l'on ne peut se permettre, lors de la conception d'une simulation, "d'oublier" quoi que ce soit, car l'introduction a posteriori du moindre paramètre allongerait sensiblement les délais.

> "Il vaut mieux retenir trop de paramètres que d'en oublier: en ajoutant quelques paramètres, et des variables de politique qui n'apparaissent pas indispensables aux clients de la simulation, nous avons beaucoup de chance de devancer leurs désirs ultérieurs et de nous économiser beaucoup d'efforts"... Mais "pour un paramètre qui se révèlera utile plus tard, nous risquons d'en introduire une dizaine qui ne seront jamais utilisés mais qui auront compliqué la simulation en pure perte".

3) Le nombre de paramètres est un aspect du problème; un autre en est leur valeur correspondant à un optimum du modèle. La simulation ne fournit la solution d'un modèle que dans des conditions déterminées; elle ne modifie pas automatiquement la valeur des paramètres pour les fixer à un niveau optimum. Cette recherche de l' optimum implique la réalisation de plusieurs simulations, chacune d'elles réalisée pour une série de valeurs données aux-dits paramètres.

4) FISHMAN apparaît assez pessimiste lorsqu'il dit qu'aucune certitude n'existe quant au fait que le temps et les efforts consacrés à la modélisation en arriveront jamais à donner des résultats utiles; l'auteur ajoute cependant que les échecs ne sont qu'occasionnels et dus à un niveau de ressources trop faible.

5) Plus spécieuse est la remarque de FISHMAN [7] lorsqu'il estime qu'un chercheur est enclin à dépeindre sa propre conception d'un problème comme la meilleure représentation de la réalité et ce, étant donné les efforts qu'il a consentis et le temps qu'il a passé à réaliser sa simulation.

1.6. ETAPES D'UNE SIMULATION.

Dans une revue générale des problèmes de simulation, une place non négligeable nous paraît devoir être réservée à l'étude des différentes étapes par lesquelles il faut passer pour atteindre les buts que l'on s'est proposés. Certes, la programmation, une des étapes obligatoires, présente un intérêt majeur mais il est bon de la replacer dans son contexte général qui est celui des étapes précitées.

1.6.1. Premier schéma.

M.G. HARTLEY [10] propose un schéma (figure 1-2) en six points pour illustrer le processus mental d'une analyse de système:

Voir Fig. 1-2

- le concepteur part d'une vue initiale du problème (a);
- il espère atteindre un but (b) par la solution qu'il proposera;
- chaque solution présentant certains aspects non satisfaisants, il est donc nécessaire de posséder des mesures ou critères d'efficacité (c);
- la simulation (d) est considérée comme la seule méthode efficace pour évaluer les performances du système;
- si les résultats de la simulation sont insuffisants, le concepteur peut être conduit à envisager l'utilisation d'un autre modèle (e);
- une évaluation préliminaire du coût (f) doit être faite, à ce stade, avant de reprendre le problème dans son ensemble.

On notera avec M.G. HARTLEY que le processus de conception est itératif bien que l'absence de flèches sur le diagramme puisse être considérée comme indiquant que tout le processus ne doit pas toujours être réalisé dans l'ordre du graphique, dans le cas présent dans le sens des aiguilles d'une montre, et que d'autre part les points (d) et (f) retiennent en général le plus l'attention du concepteur.

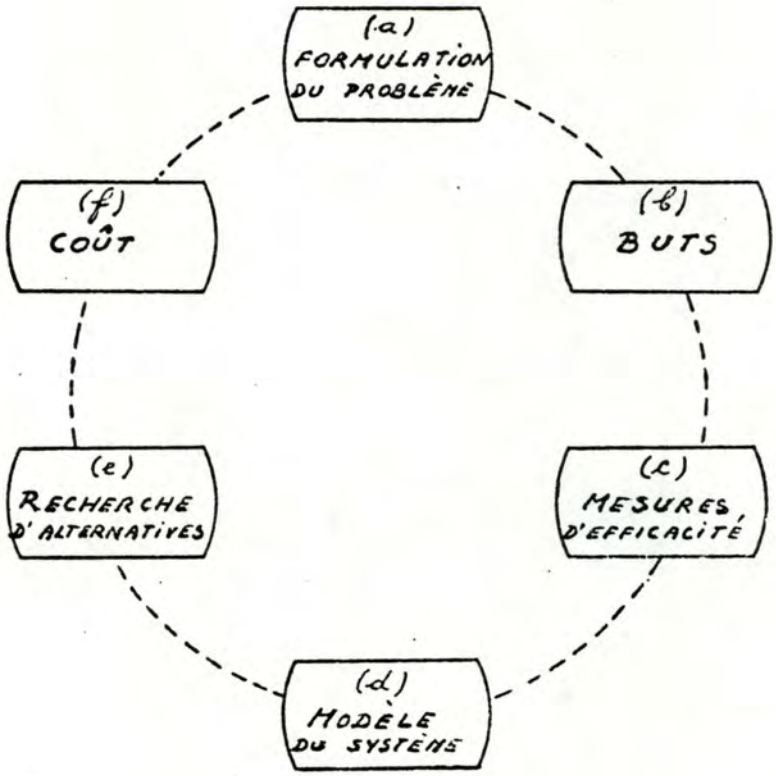


Figure 1-2: Processus mentaux d'une analyse de système [10]

1.6.2. Deuxième schéma.

De leur côté, J.G VAUCHER et P. BRATLEY [18] reconnaissent sept étapes successives tandis que Th. NAYLOR et al. en citent neuf. Les schémas de ces auteurs sont en fait très proches l'un de l'autre mais Th. NAYLOR et al. insistent davantage sur les problèmes de paramètres que nous avons abordés au paragraphe 1.5.2 /2 et 3. Nous pensons donc opportun de présenter ci-après le schéma le plus complet même si celui présenté par J.G VAUCHER et P. BRATLEY nous paraît sous-entendre de façon perceptible les étapes de NAYLOR qu'ils ne citent pas.

Remarquons avec J.G. VAUCHER et P. BRATLEY [18] que:

- la longueur-même de ces étapes souligne bien qu'un projet de

simulation est une entreprise longue et coûteuse;

- une étude de ce genre est un procédé itératif (HARTLEY mentionnait déjà ce fait: voir plus haut);
- les étapes énumérées ne suivent pas un ordre strictement sequentiel: chaque programmation nécessitera une planification expérimentale préalable;
- l'analyse des résultats peut mener à une meilleure formulation du problème;
- enfin, chaque langage de simulation implique une vision particulière de la réalité et suggère fortement un schéma de modélisation.

Le schéma proposé par Th. NAYLOR et al. [14] donné à la figure (1-3). Il nous paraît être suffisamment explicite pour nous éviter d' alourdir inutilement notre texte.

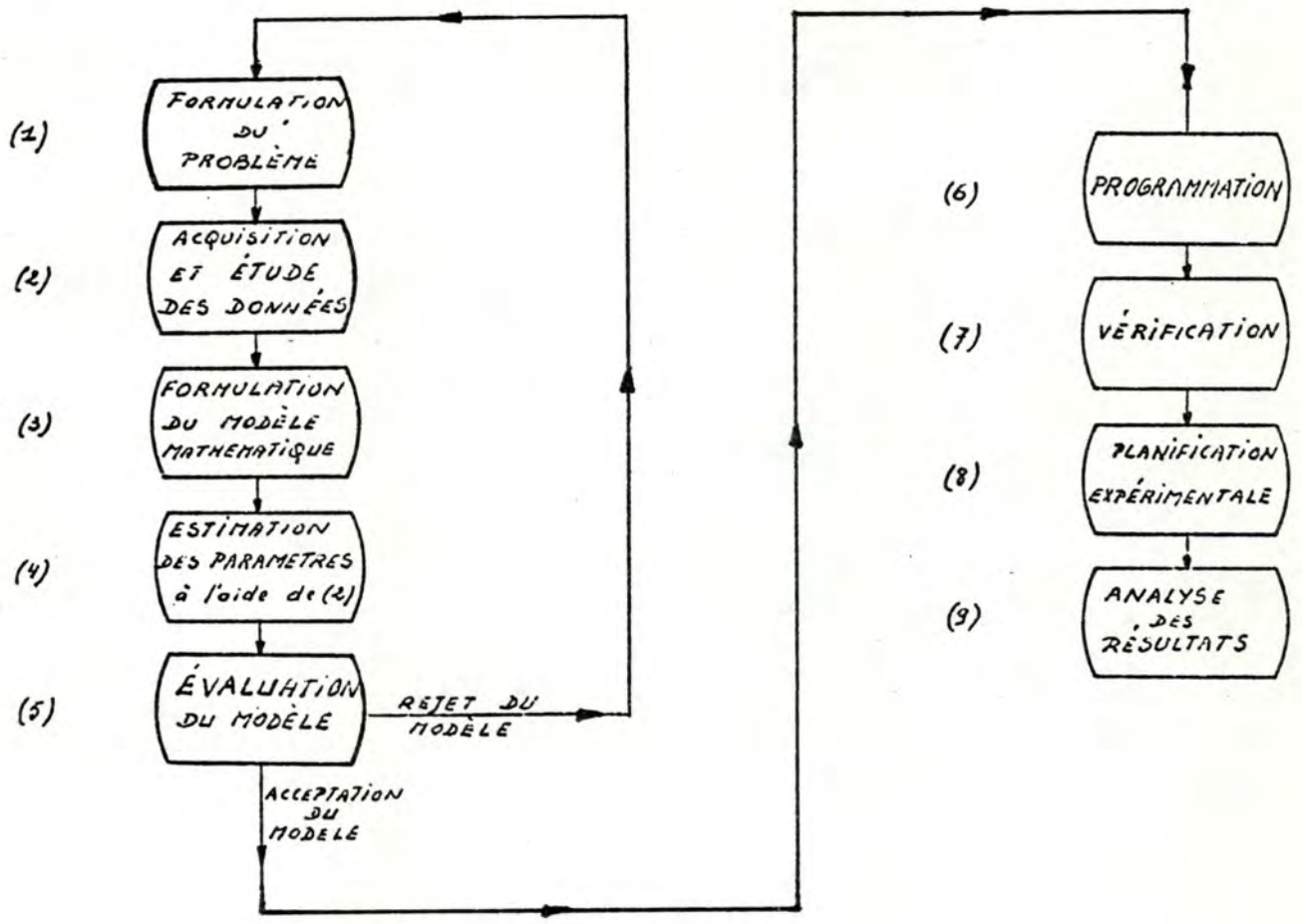


Figure 1-3: Processus mentaux d'une analyse de système [14]

CHAPITRE 2

ÉLÉMENTS DE LA SIMULATION DIGITALE A ÉVÉNEMENTS DISCRETS

2. ELEMENTS DE LA SIMULATION DIGITALE A EVENEMENTS DISCRETS.

2.1. INTRODUCTION.

Au cours de ce chapitre, nous allons préciser quelques éléments, parmi les plus importants, de la simulation digitale à événements discrets.

a) Nous abordons d'abord (par. 2.2) la base sur laquelle repose cette simulation, c'est-à-dire en d'autres termes, la représentation du comportement d'un système. Celle-ci utilise la méthode de "l'événement suivant", méthode qu'il est bon de caractériser en quelques points.

b) Nous étudions alors plus directement le problème posé et décrivons d'abord trois méthodes qui permettent de réaliser ce type de simulation; elles constituent le premier élément de la simulation.

- A ce niveau, un problème de langue se pose: la littérature dont nous disposons est surtout d'expression anglaise [7], [10], [14].
- Comme nous ne croyons pas opportun de faire un emploi systématique de mots anglais, (même si nous n'en connaissons pas d'équivalent valable en français), nous avons donc suivi VAUCHER [18] et utilisé notamment "céduler" comme correspondant du verbe "to schedule". A ce niveau, nous avons été contraints de forger le substantif "cédulation" dérivant de "céduler". Nous espérons que les linguistes ne nous reprocheront pas cette audace.

c) Nous développons ensuite de façon assez détaillée et à titre d'exemple le cas d'une file d'attente à un serveur dont le traitement se prête particulièrement bien à la simulation digitale à evénements dicrets et illustre, avec éloquence selon nous, les différences fondamentales qui existent et les possibilités des trois méthodes proposées. D'où le long développement du par. 2.4, d'autant plus que, à deux reprises, nous essayons de comparer entre elles les simulations effectuées sur ces méthodes.

d) Comme nous venons de le dire, les trois méthodes décrites constituent le premier des éléments de la simulation. Un second élément est representé par l'aléatoire: il est donc bon de rappeler quelques données fondamentales le concernant (par. 2.5).

e) Un autre élément de la simulation digitale à événements discrets consiste en la gestion du temps simulé; cette gestion s'effectue grâce au contrôleur de la simulation. Nous lui consacrons quelques pages (par. 2.6).

f) Une fois qu'on a franchi les étapes citées plus haut, on peut étudier le problème du langage qui permet la simulation à événements dicrets. Des trois types connus (langages d'assemblage, langages généraux, langages de simulation) quel est celui qui s'adapte le mieux au problème que nous étudions ? Pour répondre à cette question, il est bon de les passer en revue (par. 2.7). Une comparaison de leurs possibilités nous permet d'ailleurs de situer le problème que nous devons personnellement envisager.

g) Enfin, il nous semble utile de présenter (par. 2.8) un bref résumé de deux langages de simulation (GPSS et SIMULA) car notre implémentation s'en inspire. Le Lecteur comprendra que nous ne puissions étendre cette présentation à d'autres langages dont nous ne faisons aucun usage spécial dans notre travail; en effet, ceci alourdirait la partie bibliographique sans bénéfice aucun pour la compréhension de SIMUFOR décrit aux chapitres 3 et suivants.

2.2. REPRESENTATION DU COMPORTEMENT D'UN SYSTEME.

2.2.1. Notes préliminaires.

Nous avons vu qu'en pratique (cf. par. 1.2.2), il existe deux types de systèmes: ceux à événements continus (les changements d'état des entités y apparaissent de façon continue) et ceux à événements discrets (les changements y apparaissent de façon discrète). Il faut noter que

- les méthodes d'étude de ces systèmes (discrets et continus) sont en principe différentes
- mais aussi que certains systèmes à événements continus peuvent être simulés par un modèle à événements discrets.

L'étude des modèles à événements dicrets que nous proposons ci-après permettra donc d'élargir dans certains cas, son champ d'application logique aux systèmes à événements continus.

Rappelons enfin que, dans la simulation digitale à événements discrets, les changements d'état peuvent être représentés par un ensemble d'événements discrets et que ce sont les intervalles entre les événements qui vont déterminer la technique même de modélisation. Ces intervalles de temps peuvent être aléatoires ou déterministes, selon la loi d'apparition de ces événements.

Pour représenter un système à événements discrets et à intervalles aléatoires entre événements, on utilise le plus souvent la méthode dite de "l'EVENEMENT SUIVANT" ("next event approach").

2.2.2. Méthode de l'événement suivant.

Précisons en quelques mots en quoi consiste la méthode de "l'événement suivant" [7]:

1. le changement d'état du système a lieu lorsqu'un événement apparaît: entre deux événements successifs, l'état des entités reste constant;
2. il n'est pas nécessaire de considérer les stades intermédiaires entre deux événements puisque ce n'est que lors de l'événement suivant qu'auront lieu de nouvelles modifications d'état;

3. lorsqu'ont été considérés tous les changements d'état correspondant à un événement particulier (temps présent), le temps simulé progresse d'un seul bond jusqu'à celui de l'événement suivant (d'où le nom de la méthode);

4. tous les changements d'état possibles sont à nouveau considérés à ce stade;

5. le programme aborde une nouvelle fois l'événement suivant et le procédé se poursuit.

Par cette méthode et pour reprendre les termes propres à FISHMAN [7], la simulation est capable d'éviter ("sauter au-dessus") des temps "inactifs" alors que, dans la réalité, il est impossible de ne pas les "subir".

2.3. CONSTRUCTION D'UNE SIMULATION A EVENEMENTS DISCRETS.

Avant d'aborder la façon de construire une simulation digitale à événements discrets par la méthode de "l'événement suivant", précisons la signification générale de trois concepts:

- un processus est une suite d'événements ordonnés dans le temps;
- une activité est un ensemble d'opérations qui transforment l'état d'une entité;
- un événement est un changement de l'état du système (par. 1.2.2 et 3) ou des entités qui le constituent (par 1.2.2).

Pour situer ces trois notions l'une par rapport à l'autre, FISHMAN [7] propose la figure ci-dessous que nous croyons pouvoir interpréter comme signifiant ce qui suit: un processus comprend un certain nombre d'activités, chacune de celles-ci étant conditionnée par une succession d'événements.

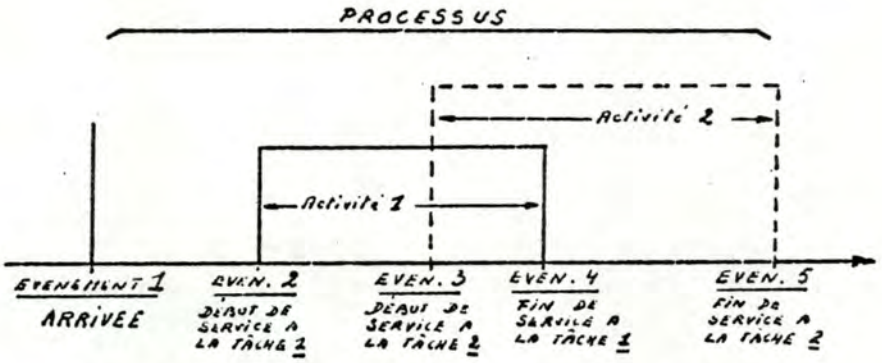


Figure 2-1: Evénement, processus, activité [7].

Ces divers concepts permettent au même auteur d'affirmer qu'il existe trois possibilités de construire des modèles à événements discrets tout en utilisant la méthode de "l'événement suivant". Nous préférons résumer ci-après les idées de FISHMAN [7] plutôt que celles de LEROUDIER [12] car elles nous paraissent plus complètes.

1. la "cédulation des événements" ("fixation d'événemnts dans le temps") ("event scheduling approach") met en valeur une description détaillée des étapes, propres à chaque événement, qui apparaissent lorsque celui-ci a lieu;
2. le "balayage des activités" ("activity scanning approach") consiste à passer en revue toutes les activités d'une simulation pour déterminer celles qui commencent et celles qui se terminent, chaque fois qu'un

événement a lieu;

3. "l'interaction des processus" (process interaction approach) attire l'attention sur la progression d'une entité à travers un système depuis son événement d'arrivée jusqu'à son événement de départ.

Nous verrons au par. 2.7 quelques exemples de langages qui utilisent ces différentes approches.

2.4. MODELE D'UNE FILE D'ATTENTE A UN SERVEUR.

Pour illustrer ces différentes possibilités de construire une simulation digitale à événements discrets, nous allons considérer l'exemple classique de ce type de simulation: la file d'attente à un serveur (Job-Shop). Dans les lignes qui suivent, nous qualifierons souvent cette file d'attente à un serveur de "demande de service à un atelier", termes qui nous paraissent correspondre à la traduction de l'expression de FISHMAN [7]: "Job-Shop".

Nous proposons ci-après la définition de la file d'attente selon FISHMAN [7] et y faisons correspondre (par des termes mis entre parenthèses) les éléments correspondants de la définition de VAUCHER [18]:

- "Dans un tel problème, une "arrivée (requête de travaux) apparaît (de façon aléatoire) et demande qu'un service (tâche demandant l'utilisation de certaines ressources) soit réalisé. Le système (atelier) répond en exécutant le service s'il le peut ou en le mettant en attente (si les ressources ne sont pas disponibles) jusqu'au moment où il parvient à l'exécuter".
- En d'autres termes, lorsqu'une demande de service apparaît, elle est satisfaite si le serveur est libre, sinon elle est placée en file d'attente. La demande et le serveur sont donc des objets ou des entités, tandis que la file est un ensemble auquel peuvent appartenir les demandes.

Dans le cas du modèle (file d'attente à un serveur) que nous considérons, les définitions présentées au par. 2.3 acquièrent un sens plus précis et parfois plus restreint:

1. un processus est une suite d'événements qui décrivent l'évolution complète d'une demande à travers l'atelier;
2. une activité est le service de la demande (sens plus restreint) (cf. par. 2.3);
3. un événement est
   - soit une arrivée ou un départ (compte tenu du fait qu'un changement d'état se produit chaque fois qu'une demande entre ou sort);
   - soit le fait que le serveur devienne libre ou occupé (ce dernier événement est cependant tributaire du premier et est donc appelé événement conditionnel);
4. l'état d'un système est le nombre de demandes enregistrées à un

moment donné (cf. par. 1.2.2).

2.4.1. "Cédulation des événements".

Sont pris en considération non seulement la cédulation des événements (qui a donné son nom à cette approche), mais aussi l'instruction conduisant à sélectionner l'événement suivant.

Voir fig. 2-2

1) "Cédulation des événements".

Un seul type d'événement est retenu (arrivée ou départ): dès qu'une arrivée apparaît, la suivante est cédulée. Il en va de même pour le départ. Chaque fois qu'un événement est cédulé, est introduit dans une liste un enregistrement qui comporte l'identification de l'événement ainsi que l'heure de son apparition.

Il eût été possible de considérer aussi comme événement (d'ailleurs conditionnel du premier) le fait que le serveur soit libre ou non. Si l'auteur ne l'a pas fait, c'est qu'il estime que cet événement est inclus dans celui qui est pris en considération et que, d'autre part, son introduction dans le modèle alourdirait ce dernier inutilement.

En effet, il serait nécessaire de créer un enregistrement, de le mettre dans une liste, puis d'opérer une recherche et une sélection supplémentaires pour le reprendre en temps voulu. Il va sans dire que l'incorporation d'événements conditionnels de ce genre occasionne des frais importants et, comme dit plus haut, inutiles parce que susceptibles d'être évités.

2)"Sélection de l'événement suivant".

Lorsque cette instruction est donnée, l'ordinateur va rechercher dans la liste des événements futurs (appelée échéancier ou SOS du terme anglais

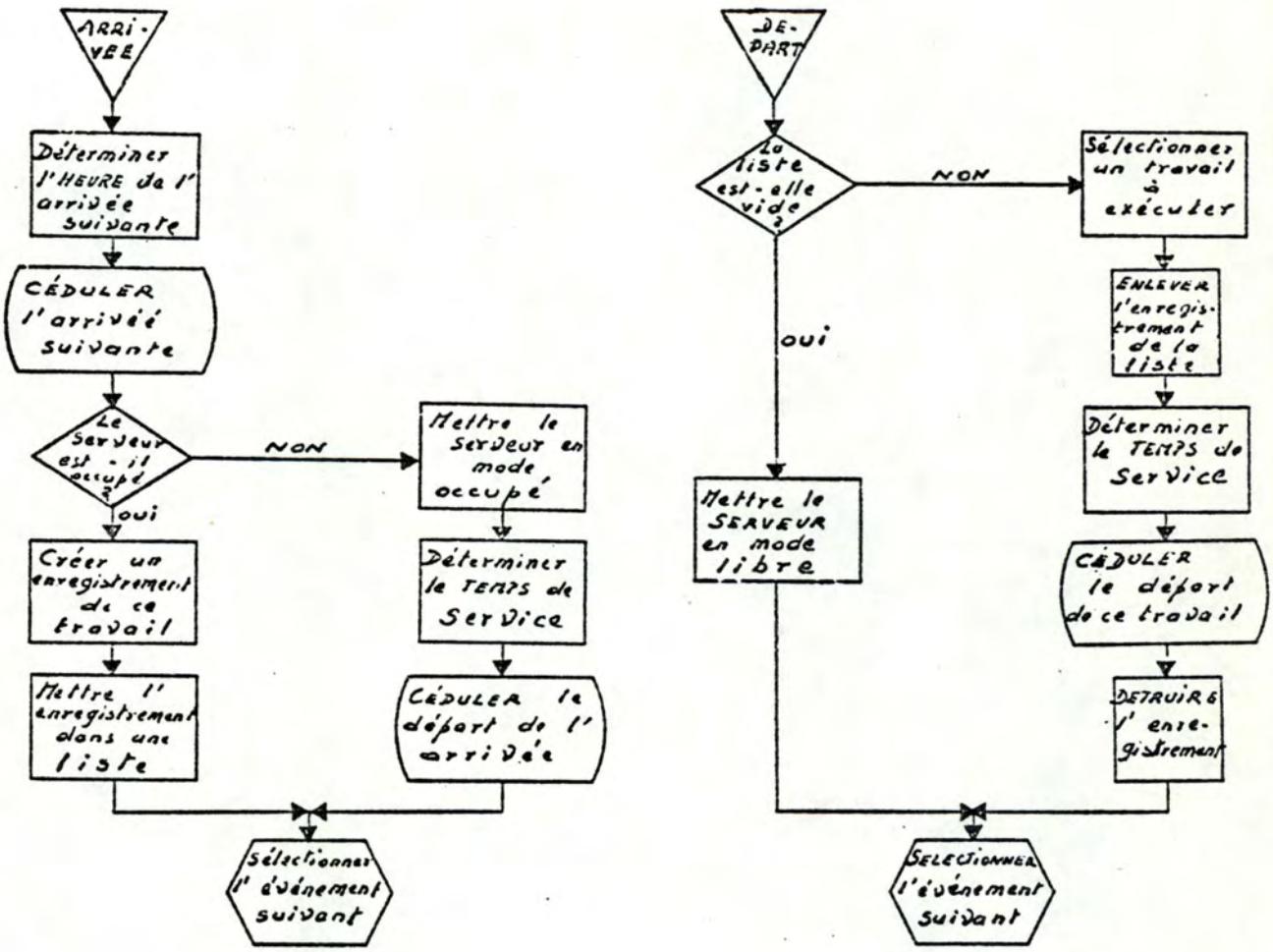


Figure 2-2: File d'attente :"cédulation des événements" [7].

"SeQuencing Set), à l'aide d'un programme appelé contrôleur de la simulation (ou routine de temps, moniteur, noyau de synchronisation cf par. 2.6), l'événement qui d'après son heure d'arrivée (et donc son heure de cédulation), doit être le premier traité. A ce moment, l'heure de simulation devient celle de cédulation: les temps morts sont donc "sautés".

Remarque:

1. Chaque arrivée possède un attribut appelé temps de service. Celui-ci peut être aléatoire ou déterministe, mais quel que soit son caractère, le modèle de simulation doit fournir un mécanisme pour le déterminer.

2. Des liens peuvent apparaître entre événements; il faut donc établir une loi de priorité parmi ceux-ci. C'est ce qu'on appelle la discipline de file. Il existe par exemple les disciplines LIFO, FIFO.

2.4.2. "Balayage des activités".

La "cédulation des événements" (cf par. 2.4.1) est basée sur la succession des événements; le "balayage des activités, lui, repose sur l'examen des activités qui apparaissent dans le problème. Il n'envisage donc pas la cédulation d'événements mais il existe forcément des événements.

Dans le cas de notre problème (une file d'attente à un serveur), il n'apparaît qu'une activité: le service d'une demande.

Voir Fig. 2-3

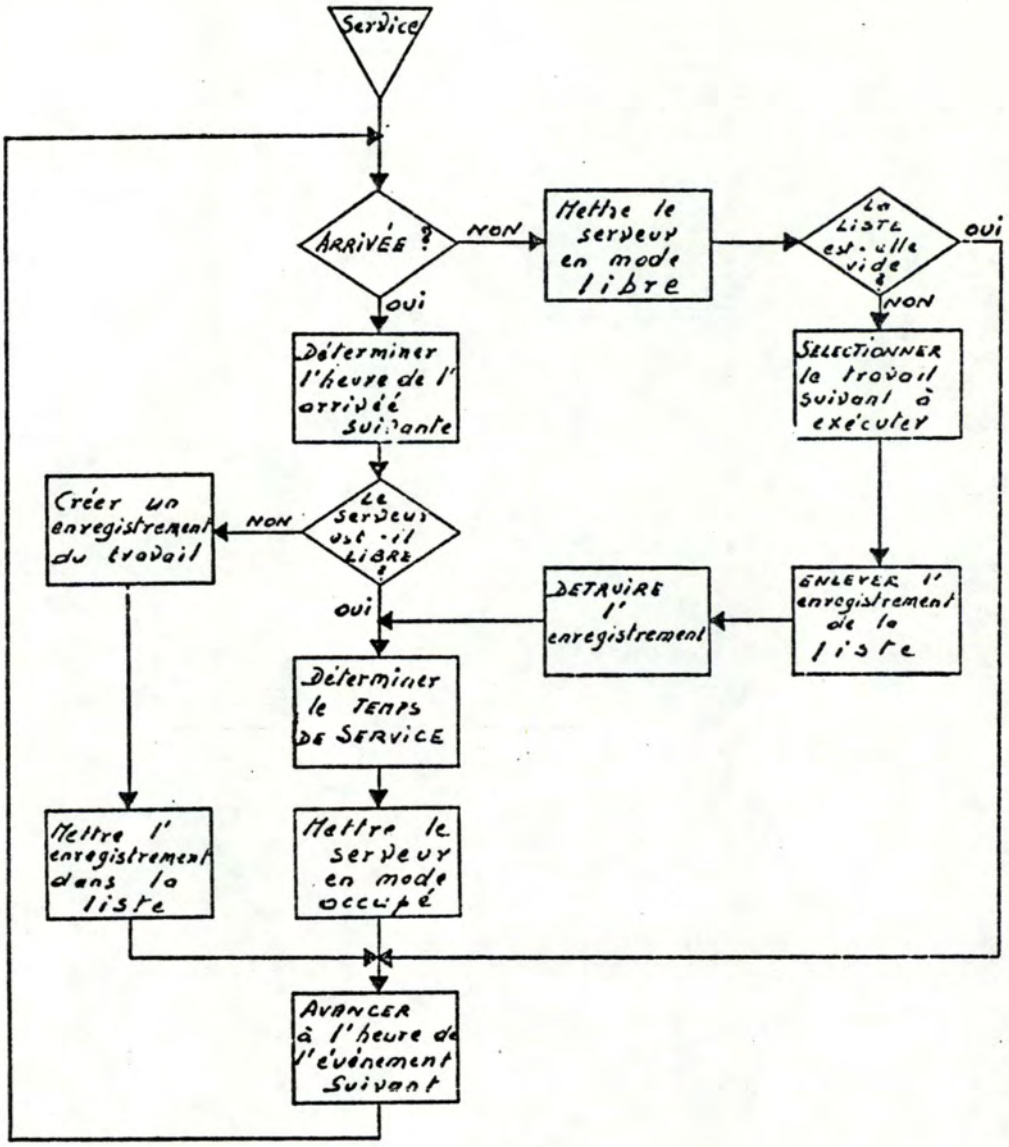


Figure 2-3: File d'attente :"balayage des activités" [7].

Le service peut débuter:

1) lors de l'apparition d'une nouvelle arrivée au moment où le serveur est libre;

2) lors de la fin du service pour autant qu'il y ait au moins une demande en attente.

Deux points importants sont à considérer:

- Lorsqu'un événement a été traité par le modèle, l'heure de simulation progresse jusqu'à celle de l'événement suivant. Pour franchir cette étape, il est nécessaire que chaque entité qui change d'état possède une horloge; ceci est vrai tant pour l'entité "arrivée" que pour l'entité "serveur". Il importe alors de déterminer si l'événement en question est une arrivée ou une sortie. Ceci est réalisé par des tests logiques.

- Comme le modèle que nous traitons ne contient qu'une seule activité, il ne permet pas de mettre cette méthode du "balayage des activités" en évidence de manière très nette (encore qu'il soit intéressant de la préciser). Ce "balayage des activités" extériorise le mieux ses avantages lorsque les activités sont nombreuses: dans ce cas, chaque fois que l'heure est avancée à celle de l'événement suivant, le programme doit balayer toutes les activités du problème pour voir celles qui peuvent être commencées ou celles qui peuvent être terminées.

2.4.3. Comparaison des deux premières approches (par. 2.4.1 et 2)

1) Les tests logiques pour déterminer si l'événement est une arrivée ou un départ, outils du "balayage des activités", remplacent la cédulation des événements et la sélection de l'événement suivant de la première approche. En effet, la "cédulation" possède un échéancier qui conserve l'identification et l'heure d'occurence des événements cédulés: le programme de contrôle de la simulation se borne à sélectionner l'événement suivant qui est devenu tête de liste.

2) L'approche "cédulation" exige que le début et la fin de chaque activité soient des événements et que ceux-ci soient cédulés. Elle s'avère donc moins intéressante que le "balayage des activités" lorsque le nombre des activités est

élevé. En effet, lorsque la liste des activités s'allonge, augmentent aussi de manière proportionnelle:

- le nombre des événements cédulés, et
- et le temps machine qui permet:

  * l'enregistrement des événements,
  * le report de ces événements dans l'échéancier,
  * leur sélection au moment opportun,
  * puis leur destruction en fin de travail.

3) Par contre, le "balayage des activités" introduit des vérifications (logiques) et celles-ci doivent être vérifiées à chaque progression du temps; ceci allonge la durée du travail.

4) Dans les cas faisant intervenir un faible nombre d'activités mais un nombre élevé d'arrivées, l'intérêt de la "cédulation des événements" par rapport au "balayage des activités" devient plus évident. En effet, le faible nombre d'activités entraîne un nombre restreint d'événements (autres qu'arrivées et départs) alors que la limitation de l'information disponible du "balayage des activités" exige, comme son nom l'indique, des balayages répétés afin que tous les changements d'état possibles puissent apparaître. La première approche bénéficie donc du peu d'événements à traiter; alors que la seconde est ralentie par cette succession indispensable de balayages.

5) Nous venons de voir que le "balayage des activités" présente un certain nombre d'avantages, malheureusement le manque de langage vraiment adéquat en interdit pratiquement l'utilisation à l'heure actuelle..!

2.4.4. "Interaction des processus".

Dans son essence, cette troisième façon d'aborder le problème de la simulation à événements discrets allie les caractéristiques un peu sophistiquées de la "cédulation des événements" à la puissance concise de modélisation du "balayage des activités".

Notons, dès à présent, que c'est cette méthode qui est à la base de notre travail personnel.

"L'interaction des processus" décrit les progrès d'une demande à travers l'atelier. Elle ne modélise donc pas les changements d'état du système. Comme les demandes arrivent à des moments différents, le comportement du système est décrit par un ensemble de processus (un pour chaque arrivée); certains d'entre eux peuvent naturellement se chevaucher.

Pour résoudre les "conflits" dus à cette possibilité de chevauchement, "l'interaction des processus" utilise des instructions comme "wait until" ou "advance" suivant que le contexte est conditionnel ou ne l'est pas. La méthode génère alors des points de réactivation dans le processus lors de la rencontre d'instructions inconditionnelles ("advance") ou d'instructions conditionnelles ("wait until") lorsque la condition n'est pas satisfaite. Les points de réactivation sont ceux, situés juste après ces instructions, auxquels la simulation retourne quand le processus est sélectionné pour un supplément d'exécution, c'est-à-dire lorsque le délai introduit par l'instruction inconditionnelle s'est écoulé ou lorsque la condition de l'instruction conditionnelle est satisfaite.

Un langage comme GPSS qui utilise l'approche de "l'interaction des processus" constitue un bon exemple de la manière dont sont traités les événements conditionnels et inconditionnels. Il possède une liste des événements cédulés (échéancier) et une liste des événements conditionnels. Après avoir exécuté tous les événements cédulés à une heure spécifiée, le programme de contrôle de GPSS (cf. par. 2.6) passe en revue la liste des événements conditionnels et exécute ceux qui peuvent l'être à ce moment. Après quoi, l'heure de simulation est avancée jusqu'à celle de l'événement devenu premier dans l'échéancier et la procédure est répétée dans son entièreté.

Les figures 2-4 et 2-5 représentent toutes deux l'interaction des processus "demande" et "serveur". La première de celles-ci tient compte du formalisme de GPSS en considérant les demandes comme actives et le serveur comme passif.

Voir Fig. 2-4 et 2-5

La seconde figure se base ,elle, sur SIMULA qui permet que les processus aient des phases actives et des phases passives. Ces deux schémas sont clairs et n'exigent, croyons nous, aucune explication particulière.

2.4.5. Comparaison entre les trois approches (par. 2.4.1-2 et 4)

1) Dans la "cédulation des événements", chaque diagramme représente un événement, alors que, dans le cas de "l'interaction des processus", un diagramme en contient plusieurs.

2) Les instructions inconditionnelles (comme "advance", "wait", ...) de l'approche "interaction" correspondent à l'esprit de la "cédulation", tandis que les directives conditionnelles (comme "wait until", ...) rappellent les vérifications logiques du "balayage des activités".

3) D'après G.S. FISHMAN [7],"l'interaction des processus" permet à l'utilisateur;

- de construire des modules d'idées, apparemment reliées, avec une plus grande facilité conceptuelle;
- de localiser la collecte des statistiques sur un seul module, au lieu de le faire au travers de plusieurs événements (ce qui a lieu dans la "cédulation des événements");
- de contrôler de façon plus aisée et plus compréhensible, l'activité de la simulation.

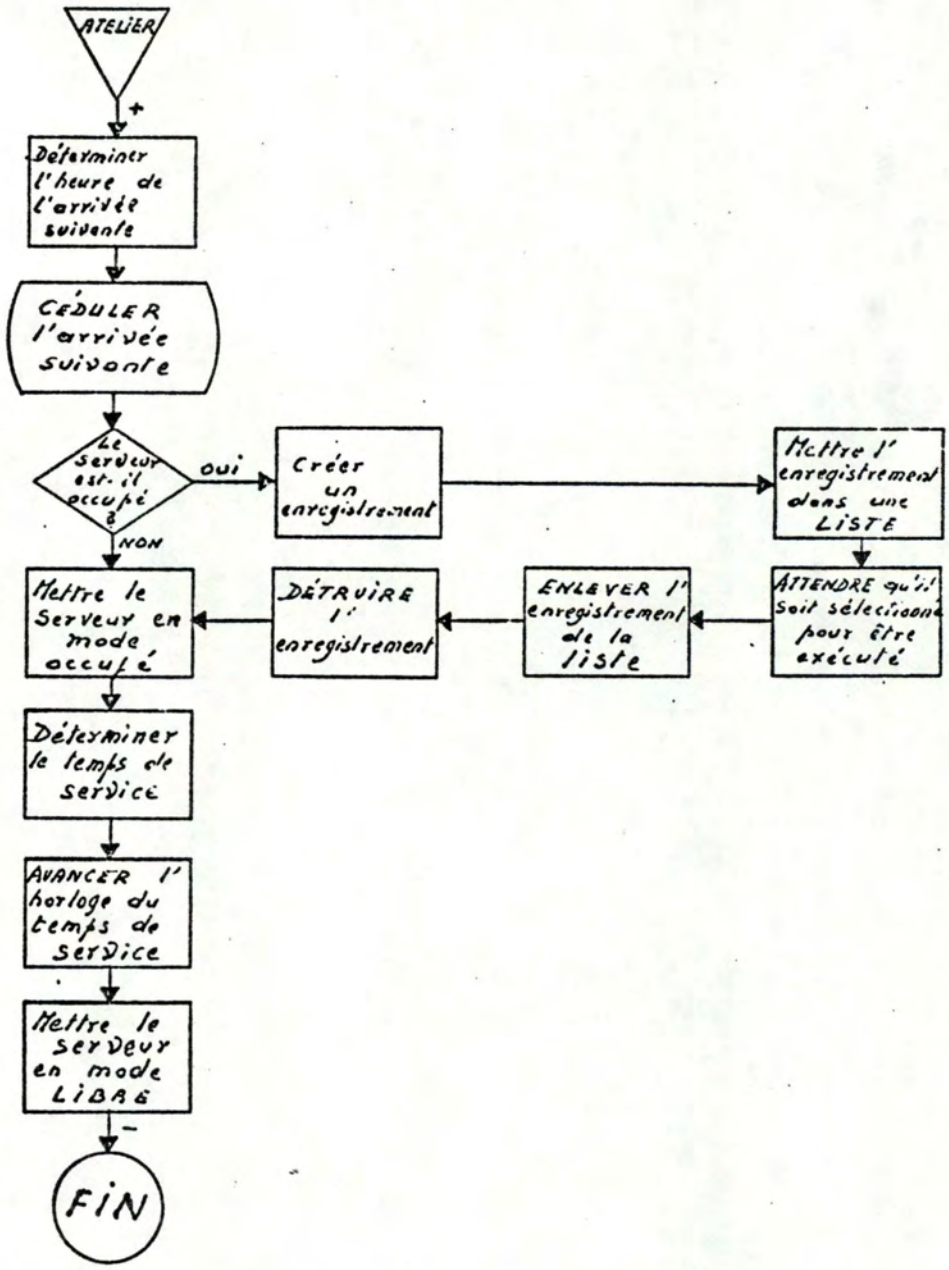


Figure 2-4: File d'attente: "interaction des proc." GPSS [7].

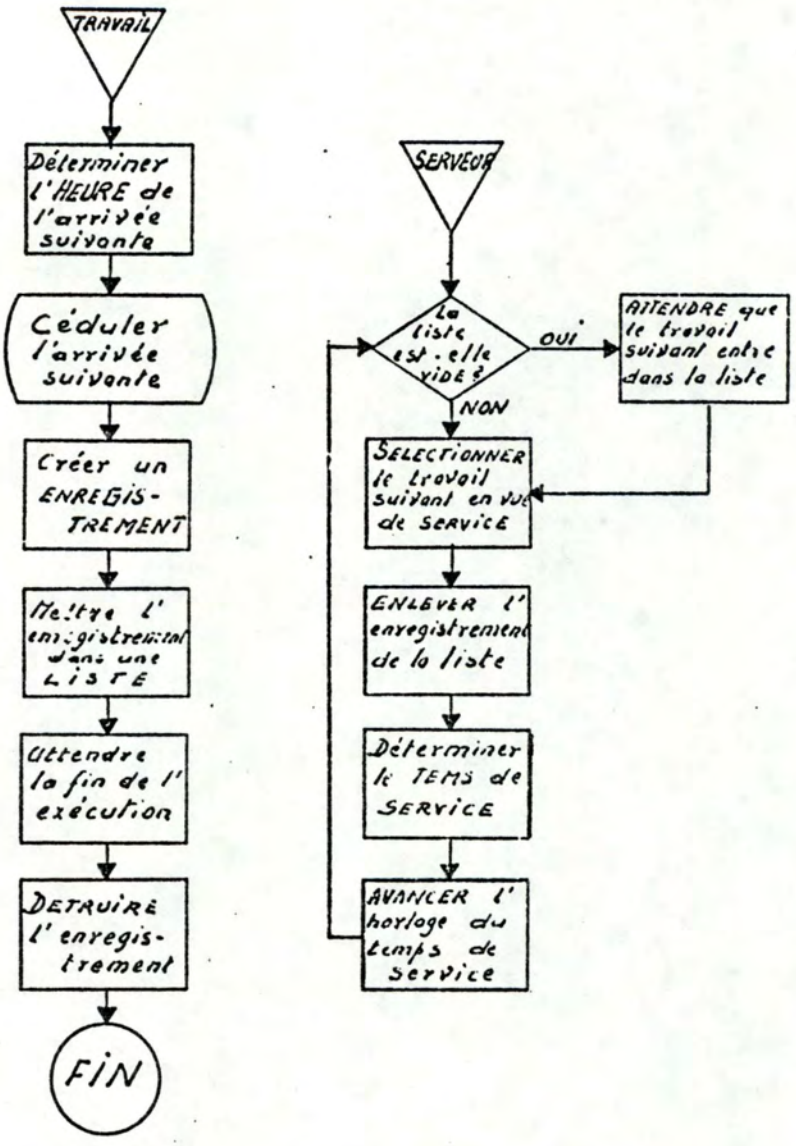


Figure 2-5: File d'attente: "interaction des proc." SIMULA [7].

2.5. SIMULATION DE L'ALEATOIRE.

Au cours de ce paragraphe, nous allons introduire des notions très importantes qui nécessiteraient des développements assez longs; ceux-ci ne nous sont cependant pas permis par les limites imposées à notre travail. Nous renvoyons donc à l'ouvrage très documenté de J. FICHEFET [6].

Pour poser le problème, disons avec cet auteur que :

> "... certains modèles de simulation nécessitent une fabrication artificielle du hasard". On est donc amené à réaliser "la fabrication artificielle du hasard uniforme c'est-à-dire la constitution d'échantillons artificiels de nombres pouvant être considérés comme des réalisations de variables aléatoires dont les valeurs possibles ont toutes la même probabilité..."

La définition "de nombre aléatoire" devient alors évidente.

De la citation ci-dessus, nous concluons que, comme la plupart de simulations sont basées sur des modèles incluant des variables stochastiques, il est nécessaire de fournir des procédés valables pour la génération de valeurs à ces mêmes variables. Il existe trois types de procédés pour générer des nombres aléatoires:

1. les méthodes manuelles;
2. les méthodes mécaniques ou électriques (qui correspondent à ce que J. FICHEFET appelle génération physique);
3. et les méthodes employant un ordinateur digital (méthodes arithmétiques de J. FICHEFET).

La technique qui est la plus utilisée et qui, en outre, doit retenir toute notre attention, est la troisième:

> "Un procédé de génération arithmétique est un algorithme créant une suite récurrente de nombres entiers et prenant les chiffres de la suite comme chiffres au hasard. En raison de la nature récurrente de la suite, il est évidemment impossible d'obtenir des chiffres correspondant à des résultats d'expériences mutuellement indépendantes. Cependant, certains procédés génèrent des suites de nombres que l'on peut, en pratique, considérer comme statistiquement indépendants. Les nombres générés par des procédés arithmétiques sont dits "pseudo-aléatoires". Ces procédes sont appelés des générateurs" [6].

En conclusion, les séquences de nombres pseudo-aléatoires sont répétitives et complètement prévisibles (d'où leur nom de pseudo-aléatoires).

Un procédé de génération de nombres pseudo-aléatoires est valable s'il satisfait à des tests statistiques et répond à quelques critères. On trouvera une étude très fouillée de ces tests statistiques (d'uniformité et d'indépendance) au chap II,6 de J. FICHEFET [6].

En conclusion, résumons cette étude en nous inspirant des travaux de HARTLEY [10] et FISHMAN [7] qui remarquent que:

1. les nombres aléatoires devraient être uniformément distribués;
2. chaque nombre aléatoire devrait être statistiquement indépendant de tous les autres nombres aléatoires de la séquence;
3. les séquences devraient être reproductibles;
4. les séquences ne devraient pas se répéter sur une période donnée;
5. les générateurs devraient être aussi rapides que possible au point de vue calcul;
6. le logiciel décrivant ces générateurs devrait être aussi concis que possible;
7. les nombres aléatoires devraient contenir suffisamment de chiffres pour que la génération de nombres sur l'intervalle choisi soit suffisamment dense.

Note

- Nous avons introduit le temps conditionnel dans ces sept exigences pour bien montrer qu'il n'est pas possible de les vérifier toutes simultanément. Il est donc nécessaire de trouver un compromis judicieux!
- Les nombres pseudo-aléatoires générés auront une "valeur" pratique d'autant plus grande qu'ils satisferont à un nombre plus élevé d'exigences parmi celles citées plus haut.

Terminons ces rappels à propos des nombres aléatoires et pseudo-aléatoires en citant la méthode la plus fréquemment utilisée pour les générer: celle-ci est basée sur la congruence (aussi appelée méthode des résidus) et génère des variables aléatoires uniformément distribuées sur $[0,m]$. Comme une simple transformation permet de les distribuer de manière uniforme sur $[0,1]$, cette

méthode peut par conséquent générer des variables aléatoires de distribution quelconque.

L'expression récursive est de la forme:

$$x_{n+1} = a\,x_n + c \pmod m$$

- a, c, m sont des entiers positifs,
- x_0 est la valeur initiale
- x_{n+1} est le résidu de l'équation
- $x_n \in [\,0\,,\,m\,[\quad \forall\, n$

L'équation ci-dessus est équivalente à:

$$x_{n+1} = a\,x_n + c - \left[\frac{a\,x_n + c}{m}\right] m$$

où [b] désigne le plus grand entier inférieur ou égal à b ($b \in \mathbb{R}$)

Cette méthode, ses variantes et ses transformations, sont présentées et discutées par J. FICHEFET [6].

2.6. CONTROLE DE LA SIMULATION.

2.6.1. Définition et rôle d'un programme de contrôle.

G.S FISHMAN [7] insiste sur un élément supplémentaire de la simulation digitale à événements discrets: le programme de contrôle. Nous avons déjà été amenés à en évoquer l'existence au par. 2.4.1-2 ainsi qu'au par. 2.4.4. C'est dire si ce programme est fondamental. Nous avons vu qu'on lui donnait aussi les noms de "contrôleur de la simulation", de "routine de temps", de "moniteur" ou de "noyau de synchronisation". Nous pouvons en donner la justification dans les termes de l'auteur précité:

- "Chaque programme consiste en un certain nombre de blocs de code; l'ordinateur les exécute dans l'ordre où il les rencontre; il le fait de façon séquentielle (c'est-à-dire instruction après instruction) à moins qu'une de celles-ci ne lui indique qu'il doive "sauter".
- Par contre, dans un programme de simulation, une heure est assignée à chaque bloc de code; à l'ordre logique des instructions, se substitue l'ordre temporel (c'est-à-dire un ordonnancement selon ces heures) et ce, grâce à un programme de contrôle...)" [7]. Celui-ci gère donc le temps simulé.

Tous les langages de simulation possèdent un tel programme de contrôle. Si l'utilisateur veut écrire sa simulation en un langage à usage général, il devra lui-même établir ce noyau de synchronisation, ce qui n'est pas une tâche aisée lorsque les événements apparaissent de façon aléatoire dans le temps.

2.6.2. Fonctionnement des programmes de contrôle.

Etudions avec FISHMAN [7] le fonctionnement des programmes de contrôle dans les trois méthodes de simulation cités au paragraphe 2.3. Observons immédiatement que le contrôleur travaille de façon différente dans les trois cas.

1) "Cédulation des événements".

- Chaque fois qu'un événement est cédulé, un enregistrement qui l'identifie et note son heure de cédulation, est créé et est mis dans l'échéancier. Dès que l'événement est exécuté, ce programme de

contrôle sélectionne l'événement suivant, sur base de l'heure, avance l'heure courante de la simulation, et repasse la main au bloc de code qui exécute les étapes correspondant à cet événement.

2) "Balayage des activités".

- Lorsque tous les débuts et fins des activités ont été réalisés à un moment donné en temps simulé, intervient le programme de contrôle qui passe en revue les horloges des entités du système. Il note les accroissements de temps que comporte chaque horloge par rapport à l'heure présente, sélectionne le plus faible de ceux-ci, avance d'autant l'horloge générale et débite du même laps de temps chacune des autres horloges. Le programme de simulation essaye alors d'exécuter les activités qui sont fixées à cette heure.

3) "Interaction des processus".

- Lorsqu'une entité entre dans le système, le programme de simulation tente d'exécuter le plus grand nombre de ses étapes. Si une période de temps simulé se termine durant cette exécution, le programme de contrôle est appelé; il crée un enregistrement qui contient l'identification de l'entité, l'heure à laquelle elle doit être reconsidérée, ainsi que son point de réactivation. Cet enregistrement est alors mis dans l'échéancier. Le noyau de synchronisation sélectionne ensuite le processus suivant, et met à l'heure l'horloge du système; le processus s'exécute alors à partir de son point de réactivation

- Jusqu'ici n'est abordé que le problème des événements inconditionnels (dont l'exécution ne dépend que du temps). Dans le cas de délais conditionnels, (lorsque les ressources nécessaires à franchir une étape s'avèrent indisponibles), un procédé similaire est utilisé. Le contrôleur intervient, crée de la même manière un enregistrement (celui-ci contient l'identification de l'entité et son point de réactivation) puis le met dans une liste d'attente.

Voir Fig.2-6

- Après avoir exécuté tous les événements inconditionnels cédulés à une heure donnée, le noyau de synchronisation balaye la liste des événements conditionnels, détermine ceux qui sont réalisables, exécute les opérations correspondantes, et ce, jusqu'à épuisement des événements. Le programme de contrôle avance alors l'horloge à l'heure du premier événement inconditionnel cédulé et le procédé se répète ad libitum.

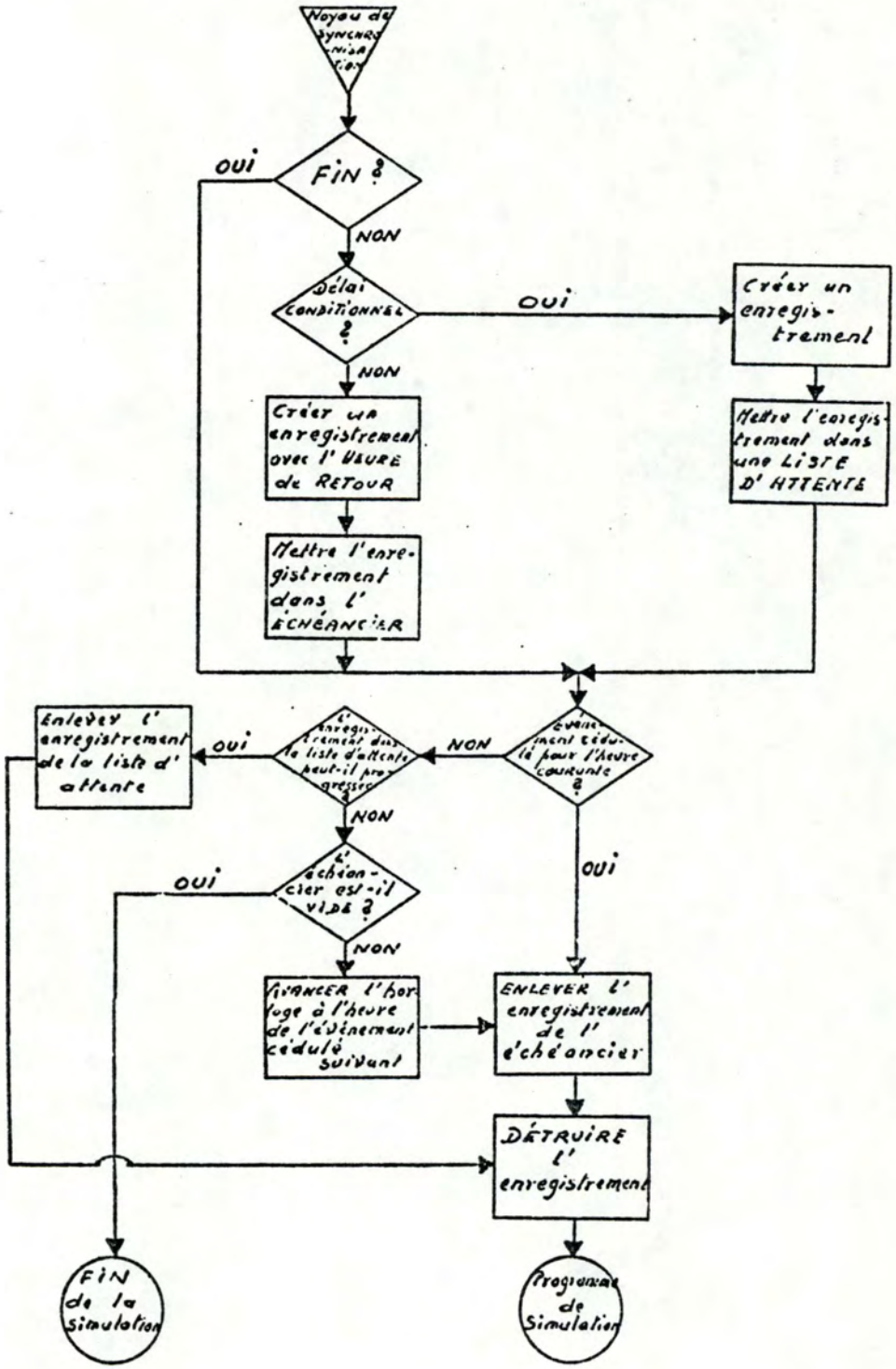


Figure 2-6: Noyau de synchronisation [7].

2.6.3. Echéancier.

L'existence et l'intérêt de l'échéancier ont déjà été notés à plusieurs reprises dans les pages qui précèdent. Il est bon de remarquer avec J.G. VAUCHER et al. [17] ainsi que J. LEROUDIER et al. [12], que c'est la gestion de l'échéancier et en particulier l'insertion (ou la réinsertion) d'un événement à une heure quelconque dans celui-ci qui constitue la plus grande activité du programme de contrôle. C'est pourquoi il est essentiel d'optimiser au maximum ce travail du noyau de synchronisation par une meilleure gestion de l'échéancier. On trouvera davantage d'information à ce sujet dans les deux références précitées.

2.7. AUXILIAIRES DE LA SIMULATION.

Une fois la simulation reconnue comme le moyen le mieux adapté à la solution d'un problème et une fois le modèle logique défini globalement, doit être envisagé le choix des auxiliaires de calcul les mieux adaptés , c'est-à-dire dans notre cas, de l'ordinateur et du langage de programmation.

Dans une entreprise ou une unité de recherche, les possibilités de choix de l'ordinateur peuvent être limitées; par contre, l'utilisateur dispose le plus souvent de plusieurs langages parmi lesquels il lui appartient de faire le choix qui lui paraît le plus approprié. Le choix de l'ordinateur ne constitue pas notre propos; nous n'aborderons donc pas ce sujet. Par contre, il nous paraît utile de nous attarder quelque peu au problème des langages, car ceux-ci jouent un rôle de premier plan, non seulement au niveau de la conception, mais aussi à celui des résultats auxquels il conduit.

Nous pouvons diviser les langages en trois catégories:

1. les "langages assembleur" ou "langages d'assemblage" appelés aussi "langages auto code" (ex: ASSEMBLEUR sur DEC, COMPASS sur CDC,...);
2. les "langages généraux" tels que les "langages scientifiques" ou "numériques" (ex: FORTRAN, ALGOL, PASCAL, ...) et les "langages universels" (ex:PL/1);
3. les "langages de simulation et de description de systèmes organisés" (ex:pour les systèmes discrets SIMULA, GPSS, SIMSCRIPT et pour les systèmes continus MIMIC).

Nous avons tenté de résumer en un tableau très simple 2-7, mais forcément approximatif de par la simplicité que nous avons voulu lui donner, les principales caractéristiques de ces trois types de langages. Il va de soi que la description que nous en faisons est loin d'être exhaustive et qu'elle est imparfaite. Remarquons que, dans le cas de langages d'assemblage, la disponibilité mentionnée à la fig. 2-7 est maximale (+++) car il existe un langage assembleur pour chaque ordinateur! Mais la portabilité d'un programme de simulation écrit en un tel langage est considérée comme nulle car son utilisation est strictement réservée aux seuls ordinateurs pour lesquels il a été conçu.

CRITÈRES	LANGAGE		
	ASSEMBLEUR	GÉNÉRAL	DE SIMULATION
PERFORMANCES DE LANGAGE	+++	++	selon les compilateurs
DISPONIBILITÉ DU LANGAGE	+++	++	+
FACILITÉ DE PROGRAMMATION et DE MODIFICATION DES PROGRAMMES DE SIMULATION	+	++	+++
ADAPTATION DU LANGAGE A LA SIMULATION	+	++	+++
LISIBILITÉ DES PROGRAMMES DE SIMULATION	0	+	+++
PORTABILITE DES PROGRAMMES DE SIMULATION	0	+++	++

Figure 2-7: Comparaison des langages de programmation.

2.7.1. Les langages d'assemblage.

Le tableau 2-7 indique que le langage machine n'est pas particulièrement adapté à la résolution de problèmes de simulation pour les raisons suivantes:

- difficultés de programmation;
- lisibilité et portabilité nulles du programme.

Ces inconvénients ne sont pas compensés par l'aspect positif de ce langage, c'est-à-dire ses performances intrinsèques: rapidité et puissance.

Dans notre travail, nous serons cependant contraints à en faire usage, mais nous le réservons aux seuls cas où le langage à usage général que nous devons utiliser (le FORTRAN) ne nous fournit pas les moyens de résoudre les problèmes qui se posent à nous. Notons pourtant que l'utilisateur éventuel de SIMUFOR ne percevra pas l'usage sporadique que nous faisons de l'ASSEMBLEUR, c'est-à-dire qu'il pourra se borner à travailler purement et simplement en FORTRAN.

2.7.2. Les langages généraux.

L'intérêt principal de l'utilisation d'un langage de programmation général réside dans le fait qu'il peut déjà être acquis préalablement par un grand nombre d'analystes-programmeurs, ce qui permettra à ces derniers de se

concentrer uniquement sur le problème qu'ils ont à résoudre, sans se préoccuper d'une nouvelle syntaxe et d'une vue différente du monde réel.

Remarque: J.W. WILLIAMS (cité par NAYLOR et al. [14]) explicite ce que nous avons évoqué au par. 2.6:

> "La difficulté majeure d'un problème de simulation est le contrôle de la séquence dans laquelle apparaissent les actions interdépendantes qui forment le modèle. Si quelqu'un s'attache à écrire un programme de simulation en n'utilisant qu'un langage de programmation à usage général, il "s'empêtrera" rapidement dans la complexité du contrôle de cette séquence; ce contrôle ne possède pas en soi un grand intérêt, et en outre il fournit un terrain étonnamment fertile au développement de petites erreurs, d'autant plus que celles-ci sont capables de produire des effets bien malaisés à mettre en évidence et sont la plupart du temps très difficiles à corriger".

LE BUT PRINCIPAL DE NOTRE TRAVAIL consiste à tenter d'améliorer la facilité de programmation, les possibilités de modification et d'adaptation ainsi que la lisibilité de programmes de simulation qui doivent être écrits en un langage scientifique (le FORTRAN) pour obtenir à ces trois niveaux, des performances équivalentes à celles des langages de simulation. Pour cela, nous proposons, en particulier, un programme de contrôle, adapté à "l'interaction des processus" et qui décharge l'utilisateur de ce travail fastidieux et complexe. Certains nous reprocheront peut-être d'avoir choisi le langage FORTRAN pour langage cible.

2.7.3. Les langages de simulation.

Les langages de simulation (comme GPSS, SIMULA, SIMSCRIPT, GASP, ...) doivent [14] et [7]:

1. produire une structure générale permettant la conception de modèles de simulation;
2. fournir un moyen rapide:
   - de convertir un modèle de simulation en un programme source;
   - de réaliser des changements dans le modèle de simulation, changements perceptibles de manière lisible dans le programme source;
3. procurer à l'utilisateur des facilités quant à l'impression et l'analyse de ses résultats;

4. au niveau des objets:

- définir les classes d'objets d'un système;
- ajuster le nombre de ces objets lorsque les conditions du système varient;
- définir les caractéristiques ou propriétés qui peuvent décrire et différencier les objets d'une même classe;
- lier les objets entre eux et à leur environnement commun.

Il existe, à l'heure actuelle, un bon nombre de langages de simulation mais il faut noter que chacun d'eux possède sa propre vue du monde réel et ne peut donc résoudre, avec la même facilité, tous les problèmes de simulation. Lors de l'étude d'un problème de simulation, il convient donc de rechercher le langage le plus adéquat (ou celui qui nécessite le moins d'étude!).

FISHMAN [7] présente les principaux langages connus au moment de la publication de son ouvrage; il les classe selon les trois approches de modélisation mentionnées au par. 2.3.

CÉDULATION des ÉVÉNEMENTS	BALAYAGE des ACTIVITÉS	INTERACTION des PROCESSUS
GASP	AS	GPSS
SEAL	CSL	NSS
SIMCOM	ESP	OPS
SIMPAC	FORSIM	SIMPL
SIMSCRIPT	GSP	SIMULA
	SILLY	SLANG
	SIMON	SOL
		SPL

Figure 2-8: Langages de simulation connus en 1973 [7].

Pour ne pas allonger notre bibliographie, nous omettons, dans ce tableau, les renvois aux auteurs des différents langages, présentés par FISHMAN.

Observons que l'on peut classer dans la troisième colonne de ce tableau:

1. le GPSSS de VAUCHER (par une extension de SIMULA, cet auteur allie les performances et les avantages de GPSS en ce qui concerne la gestion des problèmes basés sur les files d'attente);

2. le PSIM du même auteur [18];

3. le FORTSIM de BADEL et VERAN [2];

4. ainsi que le SIMUFOR faisant l'objet du présent mémoire.

Ces trois derniers sont en fait des extensions de langages scientifiques (PASCAL et FORTRAN) pour leur donner une meilleure aptitude à traiter des problèmes de simulation de files d'attente.

2.8. GPSS-SIMULA.

2.8.1. GPSS.

GPSS (General Purpose Simulation System), développé par G.GORDON d'IBM est un des plus vieux langages de simulation et peut être le plus répandu: en effet, il est disponible sur une grande variété d'ordinateurs, même différents de ceux d'IBM et son emploi n'exige aucune expérience particulière en programmation. "Ce langage est surtout destiné à la simulation de systèmes de files d'attente: c'est pourquoi les éléments primitifs du langage reflètent cette orientation" [18].

Pour en résumer les lignes directrices, nous ferons surtout appel à VAUCHER [18] et à FISHMAN [7].

1) Un programme GPSS est constitué de divers éléments: d'un côté, les objets passifs (notamment les "unités de stockage" ("storage") et les "facilités" ("facilities") qui correspondent respectivement à des ressources divisibles ou non) et d'un autre côté, les objets actifs ou mobiles appelés "transactions". Ces deux types d'objets possèdent des paramètres qui permettent de les caractériser numériquement. Dans le cas de files d'attente (destination première du langage rappelons-le), une demande de service est une transaction, un serveur simple est une facilité et un groupe de serveurs est une unité de stockage.

2) La représentation d'un système au moyen d'un modèle GPSS est un organigramme formé d'un certain nombre de blocs qui montrent le chemin qu'une transaction doit suivre dans le système: les flèches indiquent la direction à prendre et les blocs précisent les opérations à accomplir. Ces blocs sont choisis parmi une quarantaine disponible et correspondent à des déclarations qui représentent la majorité des opérations fréquemment utilisées dans une simulation. A la figure 2-9, nous donnons quelques exemples de blocs et de leur symboles.

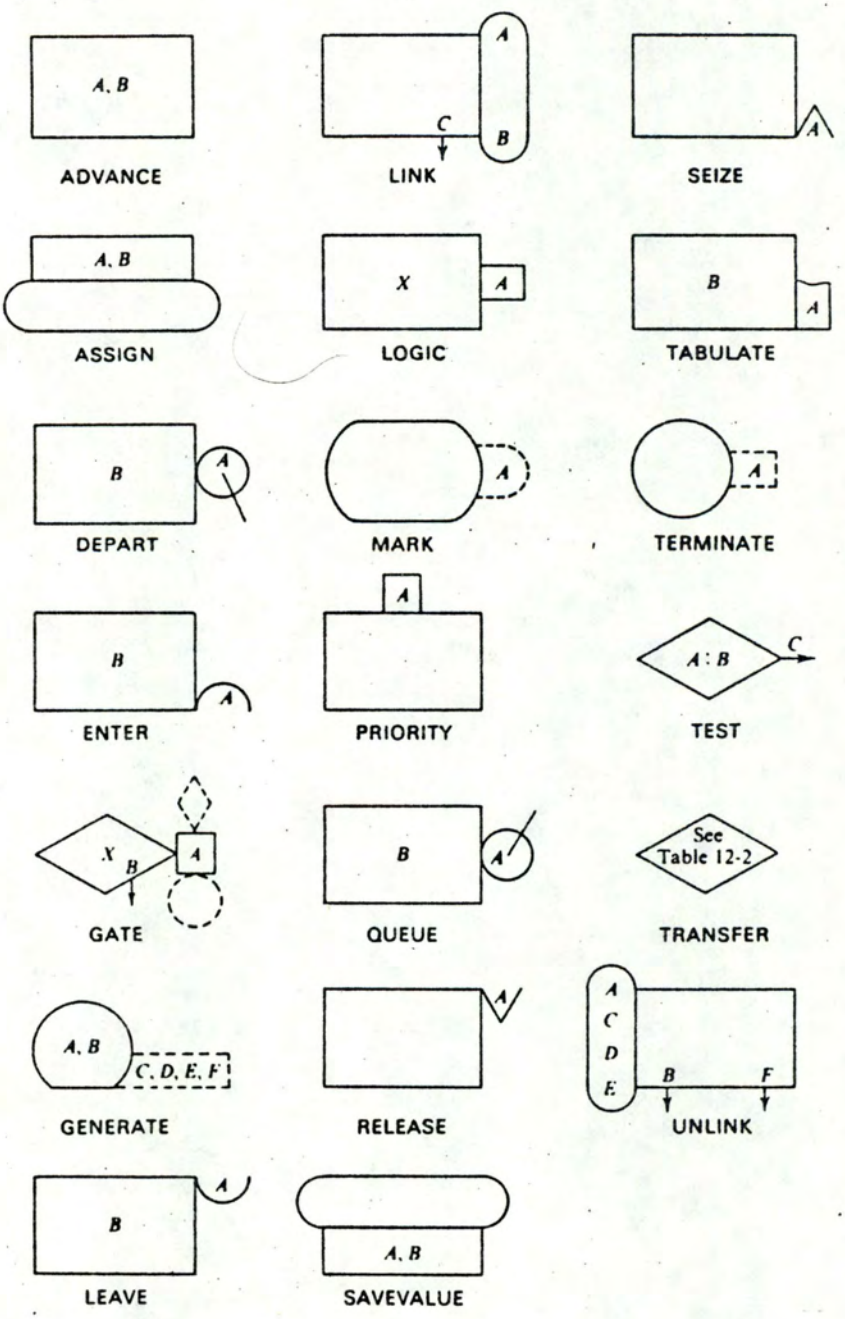


Figure 2-9: Blocs de GPSS [7]

Comme les déclarations de GPSS sont particulièrement puissantes, un petit nombre de celles-ci suffisent souvent pour conduire une simulation. Ici résident à la fois la force et la faiblesse de GPSS (cf. ci-dessous). GPSS est d'ailleurs davantage un système qu'un langage: "la majeure partie du code comprend des tables préallouées et des routines de service précompilées. Le code de l'usager est interprété pour aiguiller le contrôle à travers ces routines" [18]. D'où le minimum d'efforts exigés du programmeur, caractère cité ci-dessus.

3) GPSS utilise "l'interaction des processus" (cf. par. 2.4.4) pour organiser le comportement d'une simulation. "Au début de l'exécution d'un modèle, aucune transaction n'existe. A mesure que la simulation se déroule, des transactions sont créées à différents points de l'organigramme... Certains blocs vont retenir la transaction pour une période de temps simulé; d'autres peuvent détruire la transaction". "Certains blocs peuvent refuser d'admettre une transaction: elle doit attendre ou aller ailleurs" [18].
Voilà donc introduite la notion de temps. Celle-ci est complétée par la présence d'une Liste d'Evénememts Futurs (LEF) qui gère les événements prévus, c'est-à-dire d'un échéancier ainsi que d'une Liste d'Evénements Courants (LEC) qui gère les événements conditionnels. Un des paramètres (attributs) d'une transaction indique sa priorité; celle-ci "détermine la séquence d'exécution de deux événements cédulés pour la même heure et gouverne l'allocation des ressources aux transactions" [18].

4) "GPSS maintient automatiquement des statistiques sur l'utilisation des ressources, l'occupation des blocs et le temps passé dans certaines parties du modèle sélectionnées par l'usager. A la fin de la simulation, ces statistiques sont imprimées" [18].

5) Nous avons noté ci-dessus les avantages de GPSS, notamment la facilité avec laquelle un programmeur inexpérimenté, ou presque, peut l'utiliser. Les inconvénients [18] découlent des avantages:

- GPSS permet difficilement de spécifier des actions différentes de celles qu'il prévoit normalement;
- en outre, l'âge et la nature spécialisée du langage conduisent à des difficultés et des longueurs au niveau de l'exécution de calculs, même simples;
- d'un autre côté, chacun des blocs possède un certain nombre d'options, d'exceptions et limitations qui lui sont propres;
- enfin, et selon G.S. FISHMAN [7] le mode de base de GPSS est l'entier; l'absence de virgule flottante constitue un des points faibles du système et peut conduire à de graves erreurs.

2.8.2. SIMULA.

SIMULA a été créé au Centre de Calcul Norvégien d' Oslo en 1966 par O.J DAHL et K. NYGAARD avec la collaboration de B. MYHRHANG.

Contrairement à GPSS qui est un langage entièrement nouveau et hautement spécialisé, SIMULA est malgré son nom un langage de programmation général, basé sur l'ALGOL. Ce langage est extensible et c'est une extension standard qui en fait un langage de simulation.

"Même avec l'extension SIMULATION, SIMULA reste beaucoup moins spécialisé que GPSS. Le concept de ressources (objets passifs) n'apparaît pas; aucune collecte automatique de statistiques n'a lieu et le concept de priorité entre processus n'existe pas" [18].

J.G. VAUCHER et al. [18] montrent comment ces concepts particuliers à GPSS peuvent être programmés en SIMULA et définit une classe "GPSSS" qui permet la simulation de files d'attente avec presque autant de facilité qu'avec GPSS: il était donc intéressant de créer la classe GPSSS, basée sur SIMULA qui possède un spectre d'activité qui déborde largement des files d'attente.

SIMULA est donc un sur-ensemble d'ALGOL 60; il y apporte cependant un certain nombre de modifications. Parmi les principales modifications annoncées ci-dessus, notons ce qui suit:

1) SIMULA ajoute à ALGOL un nombre restreint de concepts essentiels à la description de systèmes complexes, dont la notion de classe: un ensemble de déclarations de données est associé à une suite d'instructions pour décrire le comportement d'un composant d'un système. Quand des valeurs numériques sont

assignées aux éléments d'une structure de données, le résultat est appelé un processus. Les processus ainsi générés peuvent conceptuellement s'exécuter en parallèle. Un nombre quelconque de processus peuvent être crées et nommés en générant des copies d'une classe.

Alors qu'un programme ALGOL spécifie une structure de données et une suite d'opérations sur des données locales au programme, SIMULA fournit un moyen qui permet de décrire, générer dynamiquement et référencer des processus; en outre, il fait en sorte que des données locales d'un processus puissent être utilisées par d'autres processus. Pour arranger les processus selon l' heure à laquelle apparaît leur activité, SIMULA fournit aussi un programme de contrôle, basé assez logiquement sur "l'interaction des processus" (cf. par. 2.4.4).

Remarquons qu'une transaction de GPSS pourrait être un type de processus. Alors que, dans GPSS, les transactions jouent un rôle actif et les unités de stockage, un rôle passif, tous les composants d'une simulation écrite en SIMULA peuvent être actifs ou passifs: chaque type de composant doit donc posséder une procédure d'activité [18].

Enfin, constatons avec VAUCHER que "le concept de classe encourage la décomposition logique d'un système complexe en sous-systèmes de taille restreinte qui peuvent être traités séparément. L'exécution quasi-parallèle des coroutines (processus) permet la modélisation parallèle des systèmes réels" [18] (la notion de coroutine a d'ailleurs déjà été annoncée à deux reprises ci-dessus).

2) "Pour permettre une description incrémentielle et hiérarchisée des systèmes, SIMULA introduit un mécanisme particulièrement utile qu'il appelle "concaténation" entre des classes générales et des classes particulières. En préfixant la description d'une classe particulière (sous-classe) par le nom de la classe générale (sur-classe), on obtient l'effet que tous les éléments de la seconde classe sont ajoutés (concaténés) à la première..." [18].

3) "C'est justement un mécanisme de préfixation qui donne un aspect extensible au langage SIMULA: bien que le langage de base ne soit pas

spécifiquement destiné à la simulation (voir plus haut), une classe standard (SIMULATION) fait partie du système" [18]. Cette classe fournit le concept de temps simulé (horloge), un échéancier, le concept de "processus" (analogue aux transactions de GPSS) introduit ci-dessus ainsi que des procédures pour céduler des événements et simuler la progression du temps.

4) "Finalement, elle donne accès à un ensemble très complet de génération de nombres aléatoires et de procédures de traitement de listes" [18], notamment par l'existence de pointeurs (variables REFerence) et l'allocation de blocs de mémoire sous contrôle du programmeur

5) Les concepteurs de SIMULA ont donc ajouté à ALGOL les quatre points repris ci-dessus. D'un autre côté, ils en ont repensé les points faibles tels que les entrées-sorties et le traitement des chaînes de caractères.

*

Un schéma donné par W.R.FRANTA [9] in [15] nous paraît résumer clairement ce qui vient d'être énoncé:

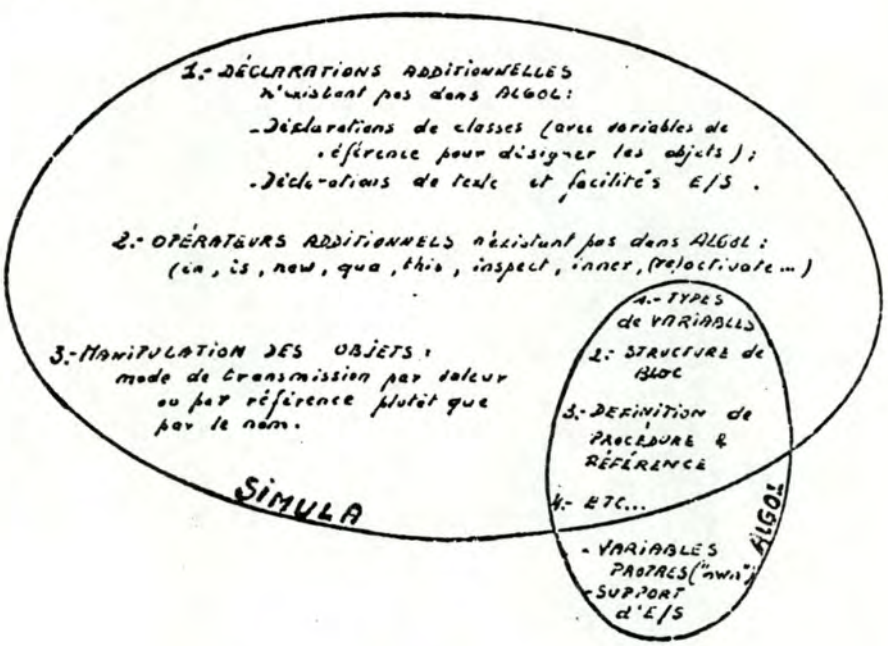


Figure 2-10: ALGOL - SIMULA

On remarque immédiatement les parties communes aux deux langages, ainsi que les éléments qui leur sont propres et par conséquent, les possibilités de chacun. Il est donc inutile de commenter davantage ce schéma.

CHAPITRE 3

UN SYSTÈME DE SIMULATION À ÉVÉNEMENTS DISCRETS EN FORTRAN

3. UN SYSTEME DE SIMULATION A EVENEMENTS DISCRETS EN FORTRAN.

3.1. L'ENJEU DE L'IMPLEMENTATION.

3.1.1. Le cadre et l'étendue de notre travail.

Lorsque l'on désire réaliser une simulation à événements discrets, sur ordinateur digital, il existe un langage parfaitement adapté à cet usage, c'est le langage SIMULA. Pourquoi alors s'exposer aux innombrables difficultés que représente la programmation d'un problème de ce type particulier dans un langage d'usage général tel que le FORTRAN, l'ALGOL, le PASCAL ou le PL/I, pour ne citer que quelques uns de ces langages? La réponse à cette question mérite d'être nuancée. Les motivations peuvent varier d'un programmeur à un autre.

Tout d'abord, il ne faut pas oublier que SIMULA, comme d'autres langages de simulation d'ailleurs, est un langage d'un type très particulier qui n'est pas disponible sur tous les ordinateurs commercialisés (surtout pas sur des "mini" ou des "micro").

D'autre part, lorsque SIMULA est disponible, l'apprentissage de ce nouveau langage représente un effort non négligeable, surtout pour un programmeur ne connaissant pas le langage ALGOL. Cet effort est d'autant plus injustifié qu'il ne s'agit parfois, pour un programmeur, que de réaliser une ou deux simulations seulement sur un sujet bien particulier.

Par contre, le FORTRAN est un langage d'usage général qui est proposé sur pratiquement toutes les gammes d'ordinateurs, y compris les plus petites. Le FORTRAN est également un langage que la grande majorité des programmeurs connaissent bien.

Nous avons donc d'une part un langage peu répandu et peu connu, le SIMULA, et d'autre part, un langage universellement (le mot n'est pas trop fort) connu, disponible et utilisé, le FORTRAN.

On peut encore ajouter que traiter une simulation dans le même langage

qu'un problème quelconque que l'on est amené à programmer offre l'avantage de permettre l'introduction d'éléments de simulation dans des programmes préexistants; et tout le monde connaît la place importante tenue par le FORTRAN dans l'ensemble des bibliothèques de programmes existant à travers le monde.

Ces différentes considérations suffisent à justifier, à notre sens, l'entreprise dans laquelle nous nous sommes lancés, à savoir, l'écriture d'un système de simulation à événements discrets en FORTRAN.

3.1.2. Le FORTRAN et ses limites.

Si le FORTRAN est l'un des langages de programmation les plus répandus, il n'est certes pas l'un des plus puissants, bien au contraire. Il fut le premier langage scientifique bien normé à être développé. Depuis, de nombreux autres langages scientifiques sont apparus, tous nettement plus puissants que FORTRAN. Ce sont les langages ALGOL, PASCAL et PL/I entre autres.

On peut essayer de dresser une liste non exhaustive, bien sûr, des lacunes de FORTRAN par rapport à ces langages plus évolués.

Le FORTRAN ne possède pas la notion de bloc bien connue de l'utilisateur d'ALGOL. En ALGOL, lorsque l'on entre dans un bloc (ou dans un sous-programme qui n'est, en fait, qu'un bloc particulier), on reçoit une copie vierge des variables déclarées à l'intérieur du bloc. A la sortie du bloc (ou à la sortie du sous-programme), cette copie des variables est rendue inaccessible au programmeur et est restituée au système qui pourra éventuellement réallouer la place-mémoire correspondante lors d'une entrée ultérieure dans un bloc. Cette technique est à la base de deux possibilités que possède ALGOL et qui feront cruellement défaut lorsqu'il s'agira d'implémenter un système de simulation en FORTRAN.

Il s'agit tout d'abord de l'allocation dynamique de mémoire réalisée automatiquement, lorsque l'on entre dans un bloc.

Et il s'agit, d'autre part, de la récursivité qui est cette propriété, que possèdent les sous-programmes ou les fonctions, de pouvoir s'appeler eux-mêmes, soit directement, soit indirectement. La récursivité est gérée automatiquement en ALGOL. De fait, chaque appel d'un sous-programme ou d'une fonction travaille sur ses propres variables, évitant ainsi l'écrasement des données d'un appel non encore terminé par celles d'un nouvel appel.

Un sous-programme ou une fonction FORTRAN ne dispose, pour toute la durée du programme, que d'un seul exemplaire de ses variables locales, ce qui interdit tout appel récursif.

Il faudra donc trouver un palliatif à l'allocation strictement statique de mémoire et à la non récursivité de FORTRAN.

Notons encore que la notion de pointeur, qui est une notion omniprésente en simulation, n'existe pas en FORTRAN, alors qu'elle existe dans un langage tel que le PASCAL par exemple.

Il faut également se méfier de la portée des variables, fort limitée en FORTRAN. En effet, en gros, sauf utilisation de l'ordre COMMON, une variable n'est accessible qu'à l'intérieur du programme, du sous-programme ou de la fonction où elle a été déclarée. Cela nous obligera à utiliser abondamment les possibilités qui nous sont offertes par l'ordre COMMON. Pour être honnête, il faut cependant noter que cette restriction sur la portée des variables en FORTRAN conduit à un élément extrêmement positif, à savoir, la possibilité de compilation séparée de chaque sous-programme ou fonction.

Force nous est donc de constater que la tâche de créer les structures et les procédures nécessaires à la simulation à événements discrets nous sera rendue plus complexe encore par la pauvreté intrinsèque des structures et des mécanismes du langage FORTRAN.

3.1.3. Notre modèle: SIMULA.

Si le langage FORTRAN se prête tel quel, par simple adjonction d'un générateur de nombres aléatoires, à ce que l'on appelle la simulation statique, il n'en est pas de même, comme nous venons de le voir, pour la simulation à événements discrets. Cette dernière met en jeu des notions, des structures de données et des types de procédures propres à cet usage. Ces éléments sont, en gros, ceux mis en oeuvre par le langage SIMULA qui nous paraît tellement bien adapté à la simulation à événements discrets qu'il va servir de base à notre modède.

Il est temps, à ce stade, de définir clairement le but que nous poursuivons. Nous désirons écrire, en FORTRAN (ou bien éventuellement dans un langage d'assemblage si l'écriture paraît difficile ou impossible en FORTRAN), toute une bibliothèque de sous-programmes et de fonctions qui seront disponibles au programmeur FORTRAN désireux d'incorporer des éléments de simulation dans certains de ses programmes, ou bien d'écrire des programmes de simulation pure en FORTRAN. Une de nos plus belles satisfactions serait qu'un programmeur habitué au langage SIMULA ne soit pas trop dépaysé si on lui demande de réécrire certains de ses programmes en FORTRAN à l'aide des sous-programmes et des fonctions que nous aurons prévus. Nous désirons donc une sémantique, ainsi qu'une syntaxe, aussi proche que possible de SIMULA, tout en restant dans le cadre de ce que permet le langage FORTRAN.

Nous appellerons notre système SIMUFOR. Bien que ce système ne soit qu'un ensemble de sous-programmes et de fonctions, et que l'écriture d'un programme de simulation à l'aide de ce système relève de la stricte programmation FORTRAN, nous dirons parfois que nous écrivons un programme de simulation en SIMUFOR. Cet abus de langage se justifie par la façon spécifique de programmer un problème de simulation malgré le cadre formel, précis et impératif du FORTRAN.

Puisque nous en sommes au chapitre des conventions, précisons également que lorsque nous parlerons de sous-programmes, et sauf mention contraire, nous ferons allusion aussi bien aux fonctions qu'aux sous-programmes. Les notions de fonction et de sous-programme sont d'ailleurs fort proches l'une de l'autre, la

seule différence étant qu'une fonction renvoie une valeur dans une variable de même nom que la fonction, ce qui permet, entre autre, l'usage d'une fonction dans une expression.

Revenons à SIMUFOR proprement dit. Comme nous l'avons indiqué, nous aimerions qu'il soit le plus proche possible de SIMULA. Si, en SIMULA, nous écrivions:

```
ACTIVATE TOTO DELAY 10 ,
```

nous aimerions pouvoir écrire en SIMUFOR, une instruction du genre:

```
CALL ACTIV (TOTO, DELAY, 10.0) .
```

De même, à la place de:

```
TOTO :- NEW PROC ,
```

nous aimerions pouvoir écrire:

```
TOTO = NEW (PROC) .
```

L'objectif final, et non dissimulé, est donc l'écriture d'un programme SIMUFOR aussi proche que possible de l'écriture d'un programme SIMULA.

3.1.4. L'objet de ce chapitre.

Dans le chapitre suivant (chap. 4), nous décrirons, d'un point de vue fonctionnel d'abord, du point de vue de l'implémentation ensuite, les différents sous-programmes et fonctions qui sont mis à la disposition de celui qui programmera un problème de simulation en SIMUFOR. Mais, avant même d'écrire ces sous-programmes et fonctions, qui représentent les opérandes de notre système SIMUFOR, nous devons nous occuper des opérateurs et mettre au point, en tenant compte des contraintes du FORTRAN, les entités et les structures de données existant dans le langage SIMULA. Cela nous amènera, dans le courant de ce chapitre, à examiner et à préciser les notions qui sont à la base du langage SIMULA et la façon de les implémenter en FORTRAN.

Nous nous emploierons ainsi, consécutivement, à essayer de trouver un palliatif aux notions suivantes:

- la portée des variables et la notation qualificative par point de SIMULA;
- la notion d'allocation dynamique de mémoire;
- la notion de classe d'objets;
- la notion de double liste chaînée;
- la notion de processus et de "co-routine";
- la notion d'événement et d'échéancier.

3.1.5. L'extension de SIMULA aux réseaux de files d'attente: le système GPSSS.

Une utilisation très fréquente de la simulation à événements discrets est constituée par la simulation de toute une série de phénomènes que l'on peut classer dans la catégorie des réseaux de files d'attente. Il n'est pas toujours facile de transposer un modèle de réseau de files d'attente dans le langage SIMULA. De même, comme nous avons voulu SIMUFOR aussi proche que possible de SIMULA, ne sera-t-il pas élémentaire de transposer un tel modèle en SIMUFOR.

Afin de faciliter la programmation d'un réseau de files d'attente, J. G. Vaucher (Université de Montréal) a ajouté une "couche de compétence" (le GPSSS [18]) à SIMULA en écrivant, en SIMULA même, une série de sous-programmes et de fonctions (il s'agit en fait d'une série de classes SIMULA pré-compilées). De la même manière, à partir des sous-programmes et des fonctions de SIMUFOR, nous écrirons d'autres sous-programmes et d'autres fonctions qui rendront la programmation de ces mêmes problèmes aussi facile qu'en GPSSS.

Nous discuterons cette extension, ou plutôt les notions de base nécessaires à cette extension, à la fin de ce chapitre.

3.1.6. Présupposés à la lecture de ce chapitre.

Nous demandons au Lecteur de ce chapitre une connaissance globale du langage SIMULA. Rappelons que les idées qui ont présidé à la conception du langage SIMULA sont présentées dans le deuxième chapitre de ce mémoire. Nous demandons au Lecteur une connaissance peut-être un peu plus précise du langage FORTRAN, car quelques problèmes d'implémentation seront résolus en faisant appel à des astuces de programmation et à certaines insuffisances bien connues des compilateurs FORTRAN. Cependant, dans la mesure du possible, nous expliciterons brièvement les notions de SIMULA et de FORTRAN mises en jeu dans les problèmes d'implémentation.

Qu'il nous soit permis de faire une autre remarque à propos du langage FORTRAN. Bien que ce langage soit régi par une norme, la plupart des constructeurs jugent opportun d'offrir à l'usager des possibilités qui ne sont pas explicitement prévues dans la norme. Certaines possibilités sont donc propres au FORTRAN d'un constructeur bien particulier. Mentionnons encore le problème des instructions d'entrées-sorties qui varient, presque par tradition, d'une firme à une autre. Notre système SIMUFOR étant destiné à un ordinateur Digital DEC 2060, et les programmes étant donc écrits en FORTRAN Digital, c'est à ce FORTRAN que nous ferons allusion. D'autre part, l'ASSEMBLEUR que nous avons dû utiliser pour certains sous-programmes et fonctions est le langage d'assemblage de Digital (le langage MACRO-20).

Pour une question de portabilité, nous avons évidemment limité au maximum le nombre de fonctions et sous-programmes écrits en ASSEMBLEUR. Certains d'entre eux n'ont pu être évités. La notion de "co-routine", par exemple, est irréalisable si l'on ne fait pas appel à l'ASSEMBLEUR.

Ces quelques remarques étant faites, nous tenons à rassurer le Lecteur: nous limiterons le plus possible, tant dans l'énoncé du problème d'implémentation que dans la rédaction des programmes, l'appel à des éléments propres au Digital DEC 2060.

3.2. DECLARATION, PORTEE ET ACCESSIBILITE DES VARIABLES.

3.2.1. Déclaration et portée des variables en FORTRAN.

Nous travaillons donc en FORTRAN! Par conséquent, oublions l'ALGOL, par exemple, où toute variable non déclarée explicitement est détectée à la compilation, et où un sous-programme a accès, non seulement aux variables déclarées dans ce sous-programme et aux arguments d'appel, mais aussi, entre autre, à toutes les variables déclarées dans les sous-programmes et le programme principal de la chaîne des appels aboutissant au sous-programme considéré.

La portée de la déclaration d'une variable est une notion assez ambiguë. On entend souvent dire que la portée d'une déclaration est globale en FORTRAN , alors qu'elle est plus limitée en ALGOL. Et c'est vrai en ce sens qu'en ALGOL, on peut, par exemple, structurer un sous-programme en une multitude de blocs possédant leurs propres déclarations de variables, alors qu'en FORTRAN, une variable, déclarée explicitement ou implicitement dans un programme ou un sous-programme, possède une portée étendue à l'entièreté du programme ou du sous-programme dans lequel elle a été déclarée. Cependant, si l'on prend un autre point de vue (qui est celui auquel nous sommes le plus sensible pour l'écriture d'un système de simulation), la portée d'une déclaration de variable peut être considérée comme plus limitée en FORTRAN qu'en ALGOL: en FORTRAN, la portée d'un nom symbolique de variable ne s'étendra en aucun cas au-delà du programme ou du sous-programme dans lequel il a été déclaré, contrairement à ce qui se passe en ALGOL.

Attachons-nous maintenant à examiner certaines particularités de la déclaration et de l'usage des variables en FORTRAN.

Ne perdons tout d'abord pas de vue que toute variable d'un sous-programme FORTRAN qui n'apparaît pas dans un ordre INTEGER, REAL, DOUBLE PRECISION, COMPLEX ou LOGICAL, est considérée comme réelle, sauf si la première lettre du symbole qui la représente est i,j,k,l,m ou n, auquel cas la variable est considérée comme entière. La plupart des variables que nous utiliserons en

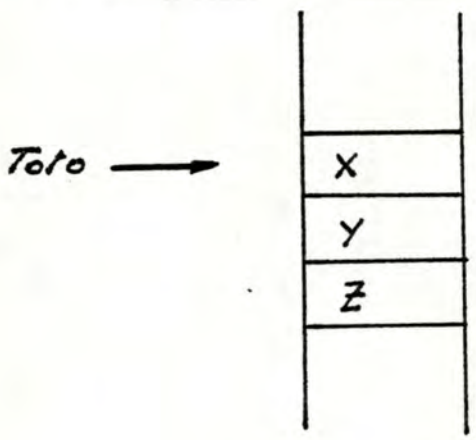
simulation sont entières. Dès lors, il serait commode que toutes les variables soient considérées comme étant entières, sauf déclarations contraires. Ceci peut être réalisé simplement par l'utilisation de l'ordre IMPLICIT INTEGER (A-Z).

Nous ne rappellerons pas systématiquement la signification et l'emploi de tous les ordres FORTRAN. Nous renvoyons pour cela le Lecteur à l'excellent livre de M. Dreyfus "FORTRAN IV" [5] ou au manuel du constructeur (Digital en l'occurrence) [8].

Qu'il nous soit cependant permis de mentionner la signification d'un ordre particulier qui occupe une place capitale dans l'implémentation de SIMUFOR. Lorsque l'on écrit la séquence suivante:

```
SUBROUTINE MACHIN
IMPLICIT INTEGER (A-Z)
COMMON /TOTO/ X,Y,Z
```

par exemple, cela signifie qu'il existera à l'exécution du programme une zone contiguë de mémoire centrale débutant à l'adresse repérée par TOTO



et d'une longueur égale à trois mots-mémoire. L'importance de l'ordre COMMON vient du fait que cette zone de la mémoire est accessible, non seulement dans le sous-programme MACHIN, mais également dans tout autre sous-programme (soit le sous-programme TRUC) où l'on écrira:

```
COMMON /TOTO/ A,B,C .
```

Il est à noter que les variables A, B et C auront les valeurs des variables X, Y et Z de MACHIN respectivement. Ce n'est pas le nom ou le symbole d'une variable

particulière qui est commun aux deux sous-programmes MACHIN et TRUC, mais le nom d'un commun particulier, c'est-à-dire le nom d'une zone contigue bien précise de la mémoire centrale. Bien que les variables X, Y, Z et A, B, C portent des noms différents, elles correspondent respectivement aux trois mêmes cases-mémoire physiques dans la mémoire centrale.

Revenons-en à la portée des déclarations de variables en FORTRAN. On peut considérer que tous les symboles sont locaux à un programme, un sous-programme ou une fonction (sauf utilisation de l'ordre EXTERNAL). Si l'on ne regarde plus les symboles, mais bien les variables "physiques" que ceux-ci représentent, nous pouvons considérer que cette règle de localité ne souffre guère que trois exceptions bien délimitées. Ce sont:

- les paramètres actuels qui sont remplacés par les paramètres formels lors de l'appel à un sous-programme ou une fonction. Le sous-programme ou la fonction s'exécute donc sur des variables définies et connues (initialisées) à l'extérieur;
- les variables déclarées dans un ordre COMMON et qui correspondent, nous l'avons vu plus haut, à des cases-mémoire qui sont en fait partagées avec d'autres sous-programmes et fonctions;
- les symboles déclarés dans un ordre EXTERNAL. Il s'agit là d'une situation très particulière où l'ordre EXTERNAL permet à un sous-programme ou une fonction d'avoir accès au nom (et donc à l'adresse) d'autres sous-programmes ou fonctions. Nous verrons plus loin en quoi cela peut nous être utile.

Nous en concluons donc que, contrairement à l'ALGOL, les seules possibilités de passer une variable physique à un sous-programme FORTRAN sont:

- d'une part, le mécanisme classique de la transmission des paramètres;
- d'autre part, l'utilisation de l'ordre COMMON auquel nous devrons très souvent faire appel.

3.2.2. Les pointeurs et la notation qualificative par point de SIMULA.

En SIMULA, outre les variables simples et les tableaux, le programmeur a la possibilité de déclarer un autre type de variables: les pointeurs. Que sont les pointeurs de SIMULA, et à quoi servent-ils? Il n'y a pas de sens à créer un

objet d'une classe donnée, au cours d'une simulation, si l'on ne se donne pas le moyen d'accéder à cet objet, pour lui appliquer certains opérateurs. Prenons quelques exemples. On sera peut-être amené à activer cet objet s'il s'agit d'un processus. On désirera peut-être le mettre dans une file d'attente ou dans une liste quelconque. Et, en tout état de cause, on devra garder la possibilité de changer la valeur de certains des attributs de l'objet que l'on vient de créer.

C'est dans ce but qu'a été créée, en SIMULA, la notion de pointeur. Le pointeur est à un objet de la simulation, ce que le symbole est à une variable, à savoir un moyen d'identifier l'objet. Notons cependant que, contrairement au lien symbole/variable (i.e. adresse) qui est réalisé à la liaison (linkage) du programme de simulation et qui reste statique durant toute la durée de la simulation, le lien entre un pointeur et un objet sera réalisé dynamiquement au cours de la simulation. Il suffit, pour s'en persuader, de noter qu'avant l'exécution du programme, aucun objet n'est créé. D'autre part, en règle générale, un objet aura une période d'activité beaucoup plus courte que la durée de la simulation. Et l'on sera souvent amené à réutiliser un pointeur pour désigner un nouvel objet, après la destruction du précédent.

En SIMULA, un pointeur est dédicacé à une classe particulière d'objets. En SIMUFOR, nous ne serons pas à même d'assurer cette restriction, sous peine d'une programmation inutilement sophistiquée.

Signalons enfin que ces pointeurs permettent l'accès aux attributs individuels d'un objet par une notation très utile, et propre à SIMULA: la notation qualificative par point. Si l'on a un objet SIMULA, repéré par le pointeur OBJ, on peut, par exemple, atteindre la variable TRUC de cet objet, par la notation OBJ.TRUC.

3.2.3. L'implémentation de la notion de pointeur en SIMUFOR.

Nous nous trouvons donc devant le premier problème de l'implémentation. Par quoi allons-nous remplacer la notion de pointeur? A ce niveau des notions de base de l'implémentation, nous désirons trouver une solution très simple qui

nous permette d'accéder aux attributs de l'objet, de façon tout à fait élémentaire.

Quel est, en FORTRAN, le type de variables qui se rapproche le plus de la notion de pointeur et qui est liée intrinsèquement à la notion d'adresse? C'est évidemment une variable entière! Une adresse n'est-elle pas une variable entière? Poursuivons notre réflexion dans ce sens. Supposons qu'un objet soit implanté en mémoire à partir de l'adresse ADD. Nous pourrions définir un pointeur vers cet objet comme étant une variable entière contenant l'adresse ADD. C'est la première implémentation de la notion de pointeur qui vient à l'esprit. Cependant, comment utiliser cette notion élémentaire de pointeur-adresse pour accéder symboliquement, dans le cadre d'un programme FORTRAN, aux attributs de l'objet repéré par le pointeur?

A ce stade, il est temps de faire une remarque complémentaire à propos du langage FORTRAN. La norme n'oblige pas le concepteur d'un compilateur FORTRAN à vérifier les dépassements de bornes lors de l'adressage d'un tableau. Ce qui signifie que, si l'on déclare un tableau par l'ordre

DIM ITAB(10)

par exemple, et si l'on tente d'adresser le centième élément de ce tableau ITAB, qui n'en comprend théoriquement que dix, aucune erreur ne sera détectée lors de l'exécution du programme, et le programmeur obtiendra, dans le format entier (puisque le symbole ITAB, commençant par la lettre I, désigne un tableau de variables entières), le contenu de la case-mémoire d'adresse X+99, si X est l'adresse où est implanté le premier élément ITAB(1) du tableau ITAB. Les réalisateurs de compilateurs FORTRAN, respectant strictement la norme du langage, ne prévoient pas, de façon standard, la vérification du dépassement de bornes d'un tableau (ils le prévoient souvent, comme chez Digital, sous la forme d'un "switch" à positionner lors de la compilation des programmes). Ce fait, qui pourrait être considéré comme une lacune du langage FORTRAN par rapport au langage ALGOL par exemple, est exploité par de nombreux programmeurs FORTRAN expérimentés pour adresser une case quelconque de l'espace-mémoire. C'est cette technique que nous allons, nous aussi, utiliser dans SIMUFOR.

Explicitons cette méthode. Grâce à l'ordre

DIM A(1)

par exemple, déclarons un tableau fictif à un seul élément, de nom A, et dont le contenu restera indéterminé tout au long de l'exécution du programme. Sa seule utilité est de permettre l'adressage de n'importe quelle case-mémoire, sous la forme d'une variable indicée. Ainsi, pour adresser la case-mémoire d'adresse ADD2, il suffit de déterminer l'indice IP à utiliser pour que l'expression (variable indicée)

A(IP)

fournisse la valeur contenue dans le mot-mémoire d'adresse ADD2.

Comme on peut le voir sur la figure 3-1, cet indice correspond au déplacement entre le mot-mémoire (supposons que ce soit le mot-mémoire d'adresse ADD1) où le compilateur a implanté le seul élément du tableau fictif A, et le mot-mémoire d'adresse ADD2, à laquelle se trouve la valeur que nous désirons adresser. Ce déplacement (correspondant à la différence ADD2 - ADD1) doit être augmenté d'une unité, pour tenir compte du fait que le premier élément d'un vecteur FORTRAN est l'élément A(1) (l'indice valant 1), et non l'élément A(0) (l'indice valant 0). Mathématiquement, l'indice devra donc avoir la valeur

IP = ADD2 - ADD1 + 1

pour que l'expression A(IP) fournisse effectivement la valeur désirée.

Nous appellerons cette technique la technique des tableaux fictifs. Elle nous permettra d'adresser l'attribut TRUC de l'objet TOTO par l'expression

TRUC(TOTO)

en lieu et place de la notation

TOTO.TRUC

en SIMULA. Avant de poursuivre notre étude dans cette voie, certaines précisions doivent encore être apportées, notamment en ce qui concerne l'allocation de mémoire aux objets de la simulation. D'autre part, les notions de classe et d'objet doivent faire l'objet d'une étude plus poussée.

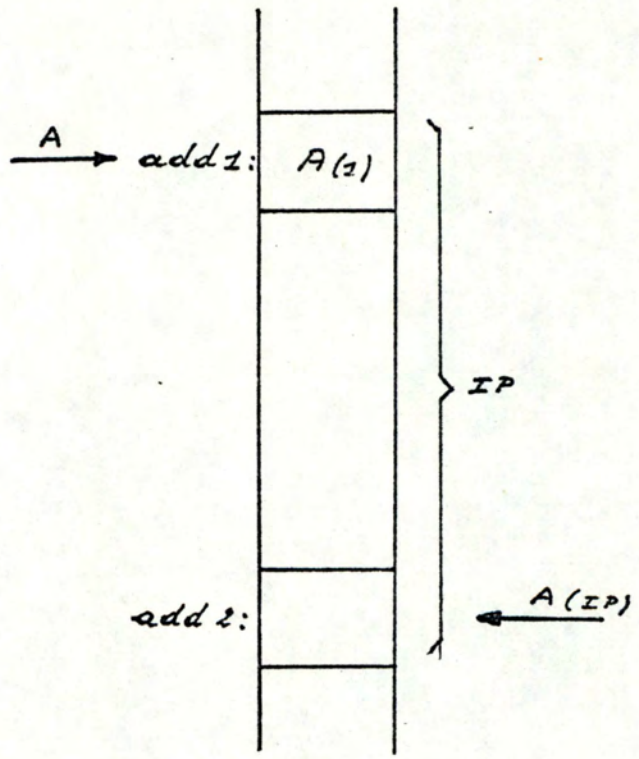


Figure 3-1: La méthode des tableaux fictifs.

3.3. ALLOCATION ET DESALLOCATION DE MEMOIRE AUX OBJETS DE LA SIMULATION.

3.3.1. Espace d'adressage du programmeur.

Le programmeur moderne travaille, presque toujours, dans un environnement de temps partagé et de mémoire virtuelle. La multi-programmation de l'unité centrale, ainsi que la gestion de la mémoire virtuelle, sont entièrement prises en charge par le système d'exploitation de l'ordinateur, de telle sorte que le programmeur peut jusqu'à en ignorer l'existence.

Les techniques de manipulation des adresses de mots-mémoire que l'on utilise pour programmer, en langage d'assemblage par exemple, se basent sur les même principes, que se soit sur les ordinateurs modernes, ou sur les premier ordinateurs (mono-programmation pure). Le programmeur dispose d'un espace de mots-mémoire (qu'il peut considérer comme contigus) numérotés de zéro au nombre de mots disponibles dans son espace d'adressage (moins un). La seule différence est constituée par le fait qu'en mono-programmation, cet espace d'adressage était la mémoire centrale elle-même, et que les adresses utilisées étaient les adresses physiques; alors qu'en multi-programmation, l'espace d'adressage est, en général, un espace virtuel; la correspondance avec les adresses physiques et la gestion des défauts de page étant gérées par le système d'exploitation de l'ordinateur.

3.3.2. Où implanter les objets de la simulation?

Nous utiliserons la méthode des tableaux fictifs pour adresser les attributs des objets créés. Cela restreint fortement les possibilités qui nous sont offertes. En effet, les tableaux fictifs et les objets de la simulation doivent être implantés dans le même espace d'adressage. D'où une limitation de place impérative: la dimension de l'espace d'adressage mis à la disposition du programmeur.

S'il n'y a pas assez de place dans l'espace d'adressage pour tous les

objets créés par la simulation, le seul remède réside dans l'utilisation de fichiers. Cependant, l'accès intensif, tant en lecture qu'en écriture, à des parties tout à fait aléatoires de ce(s) fichier(s), réclamerait un accès direct, pour ainsi dire impossible à réaliser en FORTRAN. De plus, cette méthode rendrait strictement inapplicable la technique des tableaux fictifs, et nécessiterait un adressage beaucoup trop sophistiqué.

Notons, d'autre part, qu'une destruction systématique, par le programmeur, des objets devenus inactifs, permet de limiter fortement le besoin global d'espace-mémoire d'un programme de simulation.

Pour toutes ces raisons, nous implanterons les objets de la simulation dans l'espace d'adressage même où a été chargé le programme de simulation. Deux possibilités s'offrent dès lors à nous.

La première solution consiste à définir un commun blanc très large et à y implanter les objets de la simulation. L'étendue de celui-ci doit être suffisamment grande, pour que l'on ne manque pas d'espace-mémoire; mais elle doit être, également, suffisamment limitée pour que le programme (dont la longueur a été augmentée de l'étendue du commun blanc) puisse prendre place dans l'espace d'adressage réservé au programmeur. Cette méthode est très sûre. Sauf erreur de programmation, aucune information ne viendra écraser les objets implantés dans le commun blanc. On peut cependant reprocher à cette méthode de ne pas réaliser une réelle allocation dynamique de mémoire. L'espace de mémoire nécessaire à l'implantation du commun blanc reste en effet alloué depuis le début, jusqu'à la fin, de l'exécution du programme, indépendamment du nombre d'objets créés par la simulation.

Une deuxième méthode paraît, à première vue, beaucoup plus élégante que la première. Supposons que le programme de simulation, après traitement par l'éditeur de liens, soit chargé dans l'espace d'adressage du programmeur, à partir de l'adresse zéro, jusqu'à l'adresse ADD. L'adresse ADD+1 constitue la première adresse libre de l'espace d'adressage, et l'on peut choisir d'implanter les objets de la simulation, chronologiquement, à partir de l'adresse ADD+1, jusqu'à ce que l'espace d'adressage soit complètement rempli par les objets

créés. Cette méthode réalise, elle, une véritable allocation dynamique de la mémoire. En effet, dans ce cas, on peut allouer la mémoire virtuelle paginée, page par page, au fur et à mesure des besoins de la simulation.

Malheureusement, l'analyse que nous avons faite de la place occupée par le programme FORTRAN dans l'espace d'adressage n'est pas tout à fait correcte. Aux programme principal, sous-programmes et fonctions de l'utilisateur, qui sont chargés à partir de l'adresse zéro, et à concurrence de la place nécessaire, l'exécution du programme réclame en effet le chargement de certains modules de "service", ainsi que certains tampons d'entrées-sorties, dans l'espace d'adressage. La documentation des constructeurs étant particulièrement vague à ce sujet, et les endroits de chargement de ces éléments complémentaires différant fortement d'un constructeur à un autre, aucune étude générale ne peut être réalisée à ce niveau.

En conclusion, nous dirons qu'une version portable de SIMUFOR s'appuiera sur l'usage d'un commun blanc, très large, pour implanter les objets de la simulation. Cependant, les attraits de la seconde méthode méritent que l'on cherche à surmonter les difficultés qu'elle entraîne dans le cas d'une machine particulière. Dans le cas du DEC 2060 malheureusement, tous les efforts que nous avons déployés pour réaliser un véritable mécanisme d'allocation dynamique de mémoire sont restés vains. Cependant, d'après les essais que nous avons pu mener à bien, nous avons acquis la conviction que le problème était soluble. Il suffirait pour le résoudre, de connaître l'algorithme qu'utilise le système pour allouer une nouvelle page à une tache en cours d'exécution.

3.3.3. Implantation des objets de la simulation.

Pour implanter les objets de la simulation, nous disposons donc d' une zone "contiguë" de l'espace d'adressage, dont la borne inférieure est soit la première adresse d'implantation du commun blanc, soit la première adresse libre après chargement du programme, et dont la borne supérieure est soit la dernière adresse d'implantation du commun blanc, soit la dernière adresse libre de

l'espace d'adressage. Nous implanterons séquentiellement les objets de la simulation à partir de la borne inférieure, et en incrémentant chaque fois cette dernière de la longueur de l'objet implanté, après avoir vérifié (par comparaison de la borne supérieure et de la borne inférieure) qu'il y a encore assez de place pour implanter l'objet. C'est la technique de base.

Cependant, rappelons que la durée de vie d'un objet de la simulation est, généralement, beaucoup plus courte que la durée de la simulation proprement dite, si bien que l'on peut envisager de réutiliser l'espace de mémoire alloué à un objet pour la création d'un nouvel objet. En SIMULA, un objet est considéré comme définitivement inactif lorsque le programmeur a perdu tout moyen de l'adresser, c'est-à-dire lorsque le programmeur ne dispose plus d'aucun pointeur référençant cet objet. En SIMULA toujours, il existe un mécanisme, appelé "collecteur de miettes" (ou "garbage collector") qui détecte les objets devenus inactifs, et rend l'espace de mémoire, qu'ils occupaient, libre pour l'implantation ultérieure d'autres objets.

Ce mécanisme automatique est trop compliqué à concevoir dans le cadre de SIMUFOR. La place limitée dont nous disposons pour implanter les objets de la simulation ne nous permet cependant pas de nous passer d'un système de récupération de place-mémoire. Notre "collecteur de miettes" ne sera pas automatique, mais manuel. Le programmeur devra lui-même se rendre compte de l'endroit où un objet quelconque (pointé par PT) a perdu sa raison d'être, et utiliser la procédure RETURN(PT), mise à sa disposition, pour récupérer la place qu'il occupait. Nous voulons à tout prix éviter le recompactage en mémoire des objets encore actifs. Cependant, les trous laissés en mémoire par les objets devenus inactifs ne sont pas tous de longueur égale. En effet, les objets d'une même classe ont tous la même longueur; mais celle-ci diffère d'une classe à l'autre.

La solution la plus simple, pour récupérer de la place, consiste à utiliser l'espace-mémoire occupé par un objet devenu inactif pour y implanter un nouveau de la même classe. Et c'est de cette façon que nous procéderons. Nous définirons, pour chaque classe, une liste des objets détruits de cette classe. Lorsque le programmeur décidera de détruire un objet devenu inactif, cet objet

(et donc la place qu'il occupait en mémoire) ira rejoindre la liste des objets détruits de sa classe.

D'autre part, lors de la création d'un nouvel objet d'une classe donnée, on lui octroiera la place d'un objet de la liste des objets détruits de cette classe, et c'est seulement si cette liste est vide, que l'on octroiera un espace de mémoire vierge.

3.4. CLASSES ET OBJETS.

Notons tout d'abord que la notion de sous-classe n'est pas implémentable en FORTRAN. La notion de classe, elle, occupe une place centrale dans l'implémentation de SIMUFOR.

3.4.1. Objets simples et processus.

SIMULA distingue les objets simples, les objets-liens, et les objets-processus. En SIMUFOR, n'importe quel objet peut faire partie d'une liste. Dès lors, il nous reste à distinguer les objets simples des processus.

Un objet de simulation n'a de sens que par le contenu des variables qu'il possède. Tout objet de simulation possède une série de variables que nous appellerons "variables-système" ou "attributs-système". celles-ci sont gérées par SIMUFOR et le programmeur n'a pas à en connaître l'existence. A ces attributs-système que possèdent pratiquement tous les objets d'une simulation écrite en SIMUFOR, l'utilisateur peut ajouter un certain nombre d'autres variables, à sa convenance personnelle, et pour son usage propre.

Tout objet, créé lors d'une simulation, posséde un certain nombre de variables. La différence essentielle entre un objet simple et un processus est la suivante: un objet simple n'est constitué que de variables, alors qu'à un processus est associé un code (i.e. un bout de programme) travaillant sur ces variables. Les problèmes propres aux processus seront étudiés plus tard.

Qu'il s'agisse d'un objet simple ou d'un processus, le problème du repérage de l'objet et de l'adressage de ses variables est le même.

3.4.2. La notion de classe en SIMUFOR.

La notion de classe, en SIMUFOR, est nettement moins sophistiquée que la notion de classe en SIMULA. Une classe SIMUFOR possède quatre attributs:

- un numéro qui identifie la classe considérée, et qui permet au programmeur de la repérer;
- un pointeur vers une liste des objets détruits de cette classe, afin de permettre la réalisation de notre "collecteur de miettes" manuel;
- le nombre de cases-mémoire nécessaires à l'implantation en mémoire d'un objet particulier de cette classe;
- l'adresse du début du code associé à un objet de cette classe, si cet objet est un processus.

A l'heure actuelle, le système SIMUFOR permet la gestion de vingt classes différentes. Les neuf premières sont des classes prédéfinies. L'utilité de ces neuf classes apparaîtra plus tard au cours de l'exposé. Il reste donc à l'utilisateur la possibilité de créer onze classes différentes d'objets.

3.4.3. Repérage des objets et adressage de leurs attributs.

Rappelons-nous la méthode des tableaux fictifs: c'est elle que nous allons utiliser.

Les coordonnées cartésiennes identifient univoquement un point de l'espace, à condition de se placer dans un repère bien particulier. Cette loi de la géométrie va nous aider dans la mise au point d'un système d'adressage des attributs des objets de la simulation. En effet, il paraît tentant de prendre, pour chaque classe particulière d'objets, des tableaux fictifs différents.

Malheureusement, l'indice à associer aux tableaux fictifs de la classe pour atteindre un objet particulier est identifiant de ce dernier, mais dans le cadre d'une classe particulière, et non pas dans le cadre de l'ensemble des objets de la simulation. Pour faire de cet indice un pointeur absolu vers un objet, il faut donc lui adjoindre le numéro de la classe de l'objet.

Par contre, si nous repérons tous les objets, indépendamment de la classe à laquelle ils appartiennent, par rapport à un même groupe de tableaux fictifs physiques, l'indice à utiliser pour atteindre un objet s'identifie, sémantiquement, à la notion de pointeur absolu.

La manière dont ce pointeur absolu est utilisé pour adresser les attributs d'un objet de la simulation apparaît clairement à la figure 3-2.

A ce stade cependant, il paraît nécessaire de faire un certain nombre de remarques importantes.

1) Tous les objets de la simulation seront repérés à partir du commun SYSVAR. Dès lors, comme nous l'avons déjà indiqué, l'indice d'un objet correspond, sémantiquement, à un pointeur (absolu) vers cet objet. La notion de pointeur étant plus expressive que la notion d'indice, nous parlerons, dans ce qui suit, aussi bien du pointeur d'un objet donné, que de l'indice permettant d'atteindre cet objet, par le mécanisme des tableaux fictifs.

2) Le commun SYSVAR doit être d'une longueur suffisante pour permettre d'adresser, par le mécanisme des tableaux fictifs, tous les attributs de l'objet de simulation le plus étendu.

3) La zone où sont implantés les tableaux fictifs doit correspondre à un commun pour que le programme principal, les sous-programmes ainsi que toutes les fonctions puissent atteindre les attributs de tous les objets, par la technique des tableaux fictifs.

4) Pour la beauté et la lisibilité des programmes, nous aimerions que les attributs d'un objet d'une classe particulière portent des noms parlants. Supposons que nous créions un objet de la classe rectangle, possédant deux attributs: sa longueur et sa largeur. Si le rectangle créé est repéré par le pointeur IP, nous aimerions pouvoir désigner ses deux attributs par:

LONG(IP) et LARG(IP) .

Pour cela, nous devons déclarer l'ordre

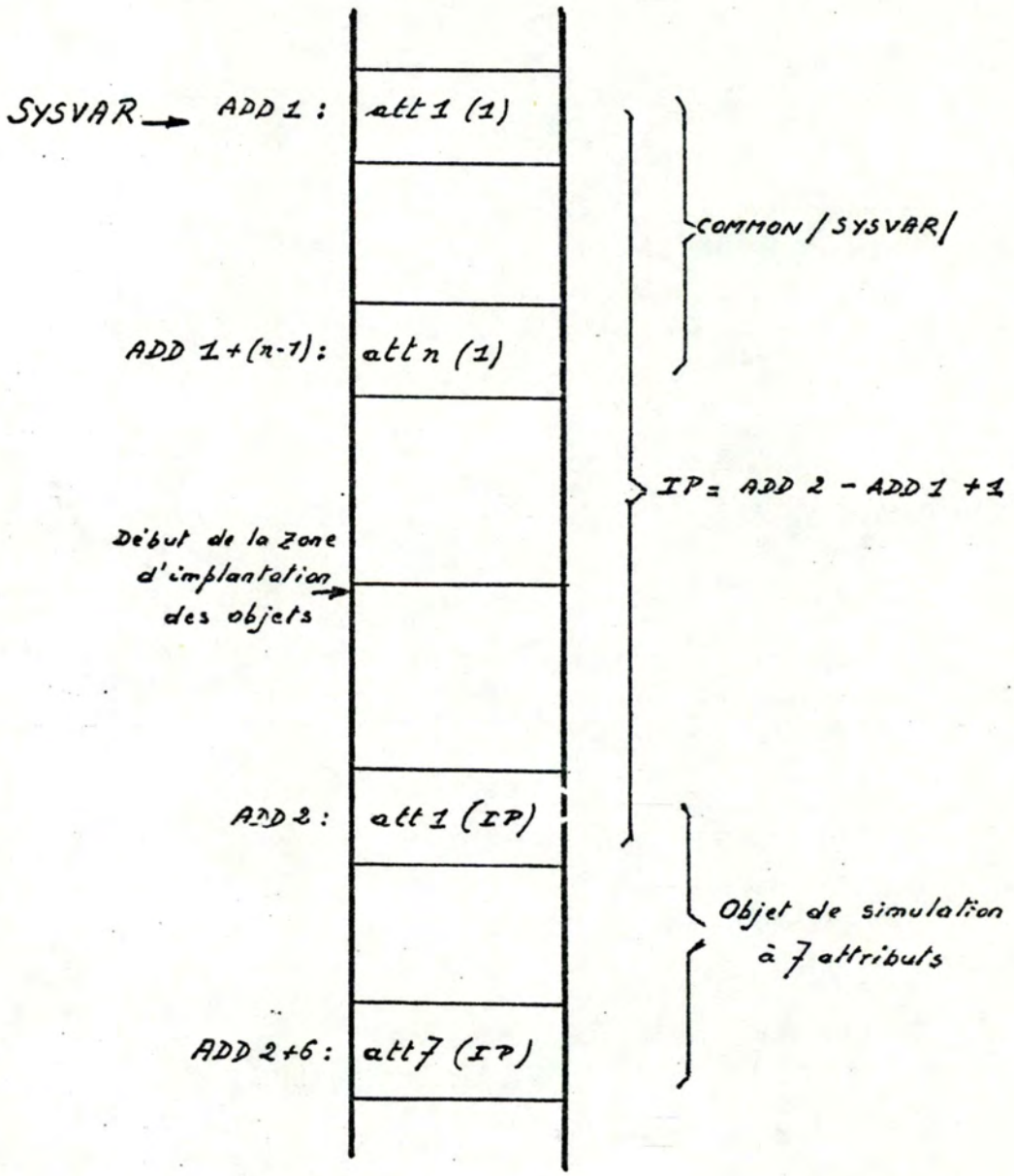


Figure 3-2: Adressage des attributs des objets.

```
COMMON /SYSVAR/ LONG(1),LARG(1) .
```

Si par contre, dans un autre sous-programme, nous avons affaire à un objet de la classe parallélipipède rectangle, identifié par le pointeur IPP, nous aimerions pouvoir désigner ses attributs par

```
LONG(IPP), HAUT(IPP) ET PROF(IPP) .
```

Pour cela, nous devons déclarer l'ordre

```
COMMON /SYSVAR/ LONG(1),HAUT(1),PROF(1) .
```

Si nous désirons adresser ces deux objets dans un même sous-programme, nous nous en tirons aisément en déclarant:

```
COMMON /SYSVAR/ LONG(1),HAUT(1),PROF(1)
DIM LARG(1)
EQUIVALENCE(HAUT,LARG)
```

5) Cette méthode des tableaux fictifs, basée sur un déplacement d'un certain nombre de cases-mémoire, est parfaitement efficace si l'on déclare des attributs d'objets de type réel, entier ou logique. Par contre, on évitera des attributs du type complexe ou double-précision car les doubles mots employés pour implanter ces attributs rendraient impossible la méthode des tableaux fictifs.

6) Par contre, le programmeur peut très bien déclarer, comme attribut d'un objet, un tableau de rang 1 ou supérieur. Prenons l'exemple d'un tableau de rang 1. Pour un vecteur à cinq positions, il suffirait de programmer

```
COMMON /SYSVAR/ ...TAB(1,5)...
```

Pour un objet pointé par IP, le quatrième élément du vecteur-attribut TAB s'obtiendrait par l'expression

```
TAB(IP,4) .
```

7) Nous n'avons, jusqu'à présent, parlé que du moyen d'atteindre les attributs d'un objet créé. Qu'en est-il du code, si l'objet créé est un processus? En fait, il n'existe qu'un exemplaire du code d'une classe d'objets-processus. Lors de la création d'un nouveau processus, une nouvelle version des

attributs de ce processus est créée. Mais ces attributs travailleront tous sur le même code qui n'est rien d'autre que la version compilée du modèle de processus, qui a été écrite par le programmeur.

Nous savons maintenant où, et comment, sont implantés les attributs des objets créés par la simulation. Nous connaissons le moyen d'accéder à ces attributs. Examinons à présent ce que sont les classes prédéfinies de SIMUFOR.

3.5. TRAITEMENT DE LISTES.

3.5.1. La notion de double liste chaînée.

La notion de liste occupe une place prépondérante en simulation. Il suffit de remarquer qu'une file d'attente n'est qu'un cas particulier de liste et, lorsque l'on connaît l'importance des files d'attente dans la plupart des simulations, on ne peut s'y tromper: la notion de liste doit aussi être implémentée dans SIMUFOR.

Il existe de nombreuses façons possibles d'implémenter la notion de liste. Prenons, par exemple, la liste des objets détruits d'une classe particulière. Lorsque l'on désire trouver en mémoire de la place pour un nouvel objet, peu importe l'objet particulier de la même classe dont il ira prendre la place; pour l'usage que l'on désire en faire, tous les objets détruits d'une classe sont interchangeables. Ces derniers ne sont chaînés dans une liste qu' afin de ne pas perdre leur trace. Dans ce cas, à quoi bon une implémentation compliquée de la notion de liste? Il suffit d'utiliser directement le pointeur vers la liste des objets détruits, dont nous avons déjà parlé et qui pointera vers un premier objet détruit. Dans celui-ci, nous aurons un pointeur vers un autre objet détruit, et ainsi de suite jusqu'au dernier de la liste, dans lequel la variable réservée au pointeur prendra une valeur conventionnelle de fin de liste. Remarquons cependant que cette utilisation de la notion de liste est un cas d'exception. Dans notre système, ce sera en fait le seul endroit où l'implémentation d'une liste pourra se faire de cette façon.

Les listes nécessaires à une simulation, telles que l'échéancier ou des files d'attente, méritent, elles, d'être implémentées de manière plus complexe. En effet, des opérations, elles aussi souvent assez complexes, devront être réalisées sur ces listes. Or, plus l'implémentation d'une liste est complexe, plus les procédures d'insertion et de recherche devraient être simples à écrire. Comme ces procédures seront d'usage très fréquent, il peut être intéressant d'en diminuer le temps d'exécution en augmentant quelque peu la place mémoire

réservée par l'implantation d'une liste.

Nous retiendrons l'implémentation sous forme de doubles listes chaînées, bien connues en SIMULA. Celle-ci permet le parcours d'une liste de façon simple, aussi bien en partant du dernier élément que du premier. De plus, on peut, sans grande peine, insérer un objet à la fin de la liste, de la même façon qu'au début.

En SIMUFOR, tout objet de simulation possède les deux attributs-système PRED et SUC qui, si l'objet appartient à une liste, contiennent respectivement le pointeur vers l'objet suivant et le pointeur vers l'objet précédant. PRED et SUC d'un objet isolé pointent, tous les deux, vers un objet conventionnel NONE, si cet objet n'appartient pas à une liste.

D'autre part, comme en SIMULA, il existe un objet particulier, appelé tête de liste, dont le pointeur SUC pointe vers le premier élément de la liste, et le pointeur PRED, vers le dernier. Une liste est représentée par un pointeur vers son objet "tête de liste" correspondant.

La création d'une liste correspond à la création d'une tête de liste, avec les pointeurs SUC et PRED qui pointent vers cette tête de liste.

3.5.2. La classe NONE.

La classe numéro 1 de SIMUFOR correspond à un objet de type NONE. NONE aurait pu être une valeur conventionnelle. Cependant, en SIMUFOR comme en SIMULA, NONE est la valeur conventionnelle que prend un pointeur lorsqu'il ne pointe vers rien de significatif, au point de vue de la simulation. Or, nous avons déjà souvent répété qu'un pointeur correspondait à l'indice permettant d'atteindre les attributs d'un objet particulier, par la méthode des tableaux fictifs. Alors, pourquoi ne pas maintenir la tautologie suivante: un pointeur pointe vers un objet de simulation? Dès lors, nous ferons de NONE un objet particulier, d'une classe spécifique, vers lequel pointera un pointeur lorsqu'il ne référence rien de significatif, au sens de la simulation.

Remarquons que lors de son initialisation, le système SIMUFOR crée, lui-même ,un objet de la classe NONE, qui, étant donné sa valeur conventionnelle, sera le seul et unique créé tout au long de la simulation. La création, par le programmeur, d'un second objet de la classe NONE aurait pour conséquence l'incohérence de toute la suite de la simulation.

3.5.3. La classe HEAD.

La classe prédéfinie numéro 2 permet au système SIMUFOR, comme à l'utilisateur, de créer des têtes de liste, c'est-à-dire des listes.

La création d'une liste renvoie à l'utilisateur un pointeur lui permettant d'accéder à cette nouvelle liste. Ce pointeur lui permettra d'effectuer, sur la liste qu'il vient de créer, toutes les opérations prévues par le système SIMUFOR et qui seront détaillées dans le chapitre suivant (chap. 4).

3.5.4. Restriction à l'usage des listes en SIMUFOR.

Le système SIMUFOR est très général, en ce sens que tout objet de simulation, créé par le programmeur, possède les deux attributs PRED et SUC, permettant de placer cet objet dans une liste qui aurait été créée auparavant.

Cependant, le système est également très restrictif, en ce sens que tout objet ne peut être placé que dans une liste à la fois. Cette restriction n'est pas très dramatique car, par expérience, il est très rare qu'un objet doive se trouver dans deux listes (ou plus) à la fois. Remarquons qu'il serait fort possible de lever cette restriction; mais alors, la programmation des sous-programmes de traitement de listes, que nous avons pu maintenir à un niveau vraiment élémentaire, s'en trouverait extrêmement compliquée.

3.6. PROCESSUS ET "CO-ROUTINES".

3.6.1. La composante "programme" d'un processus.

Nous avons déjà évoqué plus haut la notion de processus, qui représente un type d'objets composés non seulement de données (variables) comme tous les autres objets de simulation, mais auxquels est également associé un code, un morceau de programme.

Nous avons, jusqu'à présent, longuement parlé de données en limitant volontairement les objets de la simulation à un ensemble de variables.

Cependant, à quoi servirait une simulation si elle ne pouvait créer que des données? L'essence même de la simulation consiste à effectuer des traitements sur les données que l'on a créées! Nous pouvons même aller plus loin: la création des objets de la simulation (fussent-ils réduits à un ensemble de données) nécessite un certain traitement. Nous allons donc nous pencher tout naturellement, dans les pages qui suivent, sur la composante "programme" des processus.

Sémantiquement, la grande différence entre un processus et un autre objet quelconque de la simulation réside dans le fait que le processus est une entité active, alors que tout autre objet est une entité passive. Les données de la simulation sont bien des entités passives, au même titre que les fichiers pour un système d'exploitation, ou bien un dossier dans le cadre d'un travail de bureau. L'employé de bureau est typiquement une entité active. Dans notre vie quotidienne, nous sommes entourés de processus, d'éléments "moteurs", dont l'activité peut très souvent être décrite en français, ou dans n'importe quelle autre langue d'expression courante. L'activité des processus informatiques est, quant à elle, décrite dans un langage informatique. Quoi de plus naturel? La simulation que nous étudions se place, elle aussi, dans un cadre informatique. Dès lors, l'activité des processus de simulation sera décrite par un texte dans le langage informatique que nous avons choisi, à savoir, le FORTRAN. Nos processus de simulation correspondent donc à des morceaux de programmes, écrits

en FORTRAN.

3.6.2. "Routines" et "co-routines".

Cette notion de "morceau de programme", qui constitue une entité active travaillant sur certaines variables, correspond à première vue à la notion de sous-programme. L'exécution du processus débuterait alors par l'appel au sous-programme correspondant.

Tout cela serait parfait si les processus de simulation s'exécutaient entièrement, une fois lancés, ou bien, tout au moins, si l'exécution de tous les processus de simulation avait une "structure de parenthèses". En assimilant un appel de sous-programme (i.e. le début d'un processus), à une parenthèse ouvrante, et une sortie de sous-programme (i.e. la fin d'un processus) à une parenthèse fermante, l'exécution complète de la simulation devrait faire ressortir un strict respect de la structure traditionnelle de parenthèses. Si cette condition était respectée, la notion de processus pourrait, avantageusement, être implémentée sous la forme de sous-programmes.

Evidemment, une simulation qui respecterait cette condition n'aurait aucun sens. Le propre de la simulation est de dépasser ces contraintes, et d'offrir des possibilités de branchements entre sous-programmes, plus élargies que le mécanisme standard des ordres CALL et RETURN (ou des ordres assimilables à ceux-ci) prévus dans les langages d'usage général tels que le FORTRAN.

Il arrive fréquemment, en SIMULA, qu'un processus se "passive", c'est-à-dire, que son exécution soit suspendue, que d'autres processus s'exécutent et qu'à un certain moment, le processus "passivé" reprenne son exécution. La réalisation de telles séquences d'exécution nécessite la création de sous-programmes qui ne peuvent pas être écrits en FORTRAN. C'est là que se situe l'élément de l'implémentation qui nous oblige à faire appel à l'ASSEMBLEUR.

Les processus de SIMULA (et également ceux de SIMUFOR) sont ce que l'on appelle des "co-routines", par opposition aux "routines" qui ne sont que de

simples sous-programmes. Le nom "co-routine" est le nom donné à un morceau de programme dont l'exécution peut être interrompue pendant un certain temps, pour être reprise par la suite à l'endroit où elle avait été suspendue.

L'implémentation de cette notion de "co-routine" nécessite deux opérations qui ne peuvent pas être réalisées en faisant appel au seul FORTRAN.

1) Tout d'abord, un sous-programme doit permettre de sauver l'adresse à laquelle devra reprendre l'exécution du processus interrompu; et pour ce faire, il faudra, avant tout, déterminer l'endroit où l'on peut trouver cette adresse de retour.

2) Ensuite, un autre sous-programme devra permettre de reprendre l'exécution du processus suspendu, à l'endroit où on l'avait abandonnée, c'est-à-dire, en fait, "sauter" à l'adresse qui aura été sauvée par le sous-programme décrit plus haut.

3.6.3. L'écriture d'un processus en SIMUFOR.

En SIMULA, lorsque l'on désire décrire une classe de processus, on doit déclarer:

```
PROCESS CLASS TOTO
BEGIN
déclarations;
corps du processus;
END
```

En SIMUFOR, les déclarations de classes n'existent pas en tant que telles. Cependant, nous avons vu qu'à un processus de SIMUFOR est associé un certain code, représentant l'action du processus. Nous avons également vu que ce code était, en fait, une procédure dont l'exécution pouvait être interrompue, puis reprise par après. En FORTRAN, cette notion de "co-routine" n'existant pas, nous emploierons, pour décrire les processus, de simples sous-programmes:

```
SUBROUTINE TOTO
déclarations
corps du processus
RETURN
```

Rappelons que, pour une classe de processus, nous devons connaître l'adresse du début du code qui lui est associée. Or, pour déterminer des adresses de cases-mémoire à l'exécution d'un programme SIMUFOR, nous ne disposons que d'un seul outil: la fonction LOCF. Elle est écrite en ASSEMBLEUR, et permet au programmeur de connaître l'adresse qui correspond à un symbole donné. Cette fonction remplit deux missions différentes très importantes:

1) d'une part, elle permet de déterminer l'adresse d'implantation des tableaux fictifs nécessaires à l'adressage des variables des objets de la simulation. Pour ce faire, on détermine donc l'adresse d'implantation d'une variable dont on connaît le symbole;

2) d'autre part, cette fonction permet de déterminer le point d'entrée dans un sous-programme ou une fonction. En effet, il n'y a pas que les variables qui soient identifiées par des symboles; les procédures (en fait, leur point d'entrée) le sont également.

L'écriture des processus, sous la forme de sous-programmes, et la présence de la fonction assembleur LOCF nous permettent donc de déterminer les adresses de début des processus, dont nous avons besoin.

Rappelons enfin, une fois de plus, que FORTRAN ne dispose pas de la notion de bloc et qu'en conséquence, l'écriture des processus de SIMUFOR nécessitera une attention toute particulière. Il y aura un seul code pour tous les processus d'une classe donnée. L'exécution de tous les processus d'une même classe se réalisera donc sur ce code unique. Quelles variables utiliser pour l'écriture de ce code? Nous distinguerons deux types de variables:

- lorsque nous ferons référence à une variable de l'objet-processus en cours d'exécution, nous utiliserons le "pointeur" ACTUEL (variable du commun SYSVAR) qui, à tout instant de la simulation, pointe vers le processus courant, c'est-à-dire celui que le processeur central est en train d'exécuter;

- on peut également envisager d'utiliser des variables propres à la classe de processus toute entière. Dans ce cas, il est fortement conseillé de déclarer ces variables dans un commun propre à cette classe, pour éviter de mauvaises surprises dûes à un aller et retour des pages virtuelles (où est implanté le corps du processus) entre la mémoire principale de l'ordinateur et la mémoire secondaire (disques). Nous invitons même le programmeur à déclarer toutes ses variables (autres que les attributs proprement dits des processus) dans un commun, en se souvenant que le code est le même pour tous les processus d'une classe donnée.

Remarquons encore que l'implémentation des sous-programmes permettant de mettre en oeuvre la notion de "co-routine" en SIMUFOR, est essentiellement particulière à l'ordinateur utilisé. C'est pourquoi nous ne l'étudierons plus en précision, que dans le chapitre suivant.

3.7. LES EVENEMENTS ET L'ECHEANCIER.

3.7.1. Introduction.

Nous venons de définir les processus de SIMUFOR: des "morceaux de programme" dont l'exécution peut être interrompue à un moment donné, pour être reprise plus tard. Il nous reste à intégrer l'exécution (et la suspension) de multiples processus pour réaliser une simulation.

Il n'est pas inutile de rappeler le type de simulation que nous visons avec notre système SIMUFOR: la simulation à événements discrets par la méthode de "l'interaction de processus". A cet effet, nous renvoyons le Lecteur au chapitre 2 de ce mémoire.

3.7.2. Le temps simulé.

C'est à ce point de l'exposé qu'intervient, pour la première fois, la notion de "temps simulé". La simulation que nous désirons gérer est une simulation temporelle. En conséquence, elle s'appuye sur une échelle de temps qu'il nous appartient de définir. Le temps simulé (ou horloge de la simulation) est représenté par le contenu d'une variable globale à la simulation.

Cette variable se trouve dans un commun directement accessible par les sous-programmes et les fonctions du système SIMUFOR. Quant à eux, les utilisateurs de SIMUFOR n'ont pas accès à cette variable de façon directe. Cependant, SIMUFOR met à leur disposition un sous-programme (TEMPS) qui leur permet de lire le contenu de cette variable. De cette façon, ils peuvent réaliser certains tests concernant l'heure courante de la simulation. Ils peuvent également mémoriser cette heure courante à divers moments de la simulation, pour calculer certaines statistiques qui ne sont pas prévues, de façon standard, dans SIMUFOR. En aucun cas, ils ne peuvent modifier la valeur de cette variable "temps simulé". Ceci permet d'assurer la cohérence de la

simulation.

Comment évolue le "temps simulé" (que nous appelerons simplement "temps" par la suite) au long de la simulation? Très logiquement, le temps ne peut pas être décrémenté. D'autre part, il ne sera incrémenté que lors de l'interruption du processus courant, qui correspond (par l'appel au sous-programme RESUME) à la reprise de l'exécution du processus qui occupe la première place dans l'échéancier.

3.7.3. L'exécution séquentielle et la notion de quasi-parallélisme.

Le processeur central d'un ordinateur ne peut exécuter qu'une seule séquence d'instructions à la fois. A un niveau macroscopique, on peut avoir l'illusion de l'exécution simultanée (ou parallèle) de plusieurs processus. A un niveau microscopique cependant, en simulation comme en multi-programmation, on constate que l'exécution est purement séquentielle. A un moment donné (par rapport à l'horloge du processeur central cette fois), un seul processus se trouve en cours d'exécution. Supposons que trois processus soient cédulés pour la même heure HH. Lorsque le temps simulé aura atteint la valeur HH, ces trois processus s'exécuteront séquentiellement, et non simultanément. C'est ce que l'on appelle le quasi-parallélisme.

3.7.4. La discrétisation du temps de la simulation.

L'échelle du temps simulé n'est pas une échelle continue. Ce n'est pas une horloge qui bat à un certain rythme (comme l'horloge interne de l'ordinateur, par exemple). Bien que le temps simulé soit représenté par une variable réelle, il ne se présente pas comme une véritable dimension continue (réelle) de la simulation. L'échelle de temps ne sert qu'à placer les événements de la simulation. Seules les heures précises des événements ont une signification; le reste de l'échelle de temps est inutilisée.

Nous venons de parler d'événements. Cette notion est importante. Il nous appartient donc de la préciser. Comme la notion d'événement permet la mise en oeuvre efficace de l'interaction entre les processus de la simulation, nous allons d'abord aborder ce sujet.

3.7.5. L'interaction entre les processus.

L'interaction entre les processus de la simulation est réalisée à l'aide des sous-programmes dits "de scheduling" (ou de gestion des processus). Ceux-ci sont, sémantiquement, les mêmes que les ordres SIMULA portant le même nom. Ils sont décrits, de façon précise, dans le chapitre 4 de ce mémoire. Nous ne rappellerons ici que leurs noms:

- ACTIV(ATE)
- REACT(IVATE)
- P(RIORITY-)ACTIV(ATE)
- P(RIORITY-)REACT(IVATE)
- HOLD
- PASSIV(ATE)
- CANCEL
- WAIT
- ENDPRO

Quelles sont les caractéristiques de ces sous-programmes, par rapport à trois propriétés importantes de l'interaction des processus? A un niveau purement sémantique, parmi ces sous-programmes:

- quatre interrompent le déroulement du processus en cours. Il s'agit de HOLD, WAIT, PASSIV et ENDPRO. Par conséquent, le processus en tête de l'échéancier reprend son exécution;
- trois inhibent la reprise d'un processus qui avait été cédulée à une heure future bien déterminée. Il s'agit de CANCEL, REACT et PREACT;
- cinq cédulent la reprise de l'éxécution d'un processus à une heure future bien déterminée. Il peut s'agir du processus courant (HOLD) ou bien d'un processus quelconque (ACTIV, PACTIV, REACT et PREACT).

3.7.6. Définition et matérialisation de la notion d'événement.

Un événement correspond à la reprise de l'exécution d'un processus donné à une heure précise de l'échelle du temps simulé.

L'appel aux sous-programmes HOLD, ACTIV, PACTIV, REACT et PREACT génère donc un événement (ou, plus précisément, la création et la cédulation de cet événement).

Comment représenter cet événement dans notre système SIMUFOR? Nous introduirons la même notion que dans le langage SIMULA: la notion d'"event notice" que nous traduirons par notice d'événement. En fait, cette notion de notice d'événement est la matérialisation, au niveau de la simulation (programmation), de la notion intuitive et conceptuelle d'événement.

3.7.7. La classe "notice d'événement".

La classe prédéfinie numéro 3 de SIMUFOR permet la création des notices d'événements. Seul le système SIMUFOR peut créer et détruire des notices d'événements pour la programmation des sous-programmes de la gestion des processus ("scheduling"). L'utilisateur de SIMUFOR n'a, en aucun cas, accès à cette classe d'objets particuliers.

Une notice d'événement est un objet de simulation. Elle sera donc implantée dans la zone d'implantation dynamique des objets de simulation.

Une notice d'événement possède cinq attributs-système significatifs:

- un numéro de classe (CLASS), égal à trois;
- deux pointeurs (PRED et SUC) pour le chaînage de la notice dans l'échéancier;
- un pointeur (PT) vers le processus concerné par l'événement dont la notice est la matérialisation;

- l'heure (EVTIME), dans l'échelle du temps simulé, de l'événement (i.e. l'heure prévue de la reprise de l'exécution du processus pointé par PT).

3.7.8. Simulation ponctuelle, phase par phase.

Généralement, un processus ne s'exécute pas du début à la fin sans interruptions. C'est là que réside toute la différence entre un sous-programme classique et une "co-routine".

L'exécution d'un processus est divisée en plusieurs phases successives. D'un point de vue assez simplifié, la fin d'une phase d'un processus correspond à l'appel d'un sous-programme de "scheduling" du type HOLD, PASSIV ou ENDPRO.

Si l'on considère l'ensemble des processus d'une simulation, on peut déclarer, de façon générale, que la simulation toute entière se déroulera phase par phase. La fin effective d'une phase est concrétisée par l'appel au sous-programme RESUME. Celui-ci indique au processeur de suspendre l'exécution du processus en cours, et de reprendre l'exécution de celui qui se trouve en tête de l'échéancier. L'horloge de la simulation est également mise à l'heure lors de l'appel à RESUME.

Du point de vue de la simulation, une phase est considérée comme instantanée. Qu'est-ce que cela signifie? Supposons que l'ordre RESUME rende le contrôle au processus PROC, à l'heure simulée HE. L'exécution complète de la phase courante du processus PROC se poursuivra jusqu'à la rencontre de l'ordre RESUME. On considère que l'exécution de l'ensemble des instructions de cette phase se déroulera entièrement à l'heure HE, comme si l'horloge restait bloquée pendant la totalité de l'exécution de cette phase. En fait, l'horloge de la simulation, concrétisée par la variable TIME, garde une valeur constante lors de l'exécution d'une phase. Seul l'appel à RESUME modifie l'heure de la simulation, pour lui donner la valeur de l'heure de l'événement pris en considération (celui que l'on trouve en tête de l'échéancier).

Il nous semble opportun de faire, ici, une remarque très importante concernant la programmation des procédures de gestion des processus. Par soucis d'homogénéité, on appelle toujours RESUME à la fin d'un sous-programme de "scheduling". Cependant, dans le cas des sous-programmes (P)ACTIV, (P)REACT et CANCEL, le contrôle est évidemment rendu au sous-programme appelant. En effet, ces procédures ne délimitent pas deux phases conceptuelles différentes. Pour cela, il suffit de ne pas enlever de l'échéancier la notice d'événement du processus appelant.

3.7.9. La notion d'échéancier.

Au cours de la simulation, des événements sont cédulés. Ceci entraîne la création de notices d'événements, avec leurs deux attributs-système principaux: l'heure de l'événement et le pointeur vers le processus concerné...

Au fur et à mesure du déroulement de la simulation, ces événements seront pris en compte. Il nous faut donc les mémoriser. Pour ce faire, nous allons créer une liste dans laquelle nous lierons l'ensemble des notices d'événements créées. Cette liste constitue l'échéancier. Nous emploierons parfois l'abréviation "SQS" du terme anglais "SeQuencing Set" utilisé en SIMULA.

Comme le suggère son nom, l'échéancier est une sorte d'agenda des événements prévus dans le futur. On pourrait le comparer à un calendier à effeuiller qui posséderait les caractéristiques suivantes:

- chaque cédulation d'une phase d'un processus correspondrait à une feuille du calendrier que l'on irait intercaler entre les autres, d'après l'heure de la réactivation, par rapport à l'échelle du temps simulé;
- ce calendier ne posséderait une feuille qu'aux heures concernées par un événement;
- d'autre part, à une heure donnée pourraient éventuellement correspondre plusieurs événements (donc plusieurs feuilles consécutives).

Il s'agirait bien d'un calendrier à effeuiller. En effet, la feuille de garde correspondrait, à un moment donné, au processus courant. La fin de la phase

courante du processus courant correspondrait alors à l'arrachage de la première feuille du calendrier. La nouvelle feuille de garde, ainsi révélée, déterminerait le nouveau processus courant.

3.7.10. L'organisation de l'échéancier.

Dans l'échéancier, les notices d'événements sont donc triées par heures d'événement croissantes. Cette règle permet d'avoir constamment le processus courant en tête de la SOS. Lors de l'appel au sous-programme RESUME, après le retrait éventuel de l'ancien processus courant, le contrôle sera rendu au processus dont la notice d'événement se trouvera en tête de l'échéancier, et qui deviendra le nouveau processus courant. Cette constance de la place occupée par le processus courant, en tête de la SOS, est évidemment de nature à accélérer l'exécution du sous-programme RESUME. Par contre, elle nous oblige à maintenir l'échéancier trié sur l'heure d'événement. Cette contrainte est de nature à compliquer l'insertion d'un événement dans l'échéancier. Dès lors, pour l'organisation de l'échéancier, deux solutions s'offrent à nous.

1) SIMUFOR est doté de la notion de double liste chaînée. Pour la gestion de ce type de listes, nous disposons (cfr. chap. 4) de toutes les procédures nécessaires d'insertion, de retrait et de consultation. Nous pouvons, dès lors, organiser l'échéancier sous la forme d'une double liste chaînée (par l'appel SOS = NEW (HEAD), où SOS désigne le pointeur vers la liste-échéancier) et utiliser les procédures standards d'insertion. Cette solution séduit par sa simplicité. Cependant, le tribut à payer pour maintenir cette simplicité paraîtra lourd à certains. Il nous faudra prendre en compte des temps d'exécution, parfois assez importants, des procédures d'insertion des événements dans l'échéancier. Ce manque de performances est dû au balayage séquentiel (avant ou arrière) obligatoire pour repérer l'endroit de l'insertion.

2) C'est précisément la rapidité des insertions qui justifierait l'implémentation d'une organisation plus performante de l'échéancier. Nous pourrions utiliser un arbre ou une liste indexée [17] [12] . Cependant, un gain

de performances ne semble pas assuré dans tous les cas. Certaines organisations ne se montrent performantes que pour un certain type de simulations, ou bien à partir d'un certain nombre d'événements présents dans l'échéancier. Des mesures [17] [12] semblent indiquer qu'il n'y a pas de panacée universelle en matière d'organisation d'échéancier. Et surtout, il faudrait réécrire tout un ensemble de procédures permettant de gérer l'échéancier dans l'organisation particulière que nous aurions choisi de lui donner.

C'est la raison pour laquelle nous avons choisi, dans un premier temps de l'implémentation de SIMUFOR , d'organiser l'échéancier sous la forme d'une double liste chaînée classique. Une étude plus approfondie de la question permettrait sans doute de trouver une organisation plus performante dans la majorité des cas de simulation. L'écriture de nouvelles procédures spéciales de gestion de l'échéancier permettrait alors de réaliser un gain de performances (du point de vue du temps d'exécution des procédures d'insertion surtout).

Tel que nous l'avons implémenté, voici comment se présente l'échéancier, à un moment quelconque de la simulation.

voir figure 3-3.

3.7.11. L'usage des événements et les différents status possibles d'un processus.

A un processus ne peut correspondre, à un moment donné, qu'un seul événement au plus. En effet, un processus reste une suite séquentielle d'opérations. Si les différentes phases peuvent ne pas être exécutées consécutivement, leur exécution ne peut se chevaucher. Une phase d'un processus ne peut être cédulée que si la phase précédente a été totalement exécutée.

Par contre, plusieurs événements peuvent être cédulés pour une heure déterminée (pouvu que ces événements correspondent à des processus différents).

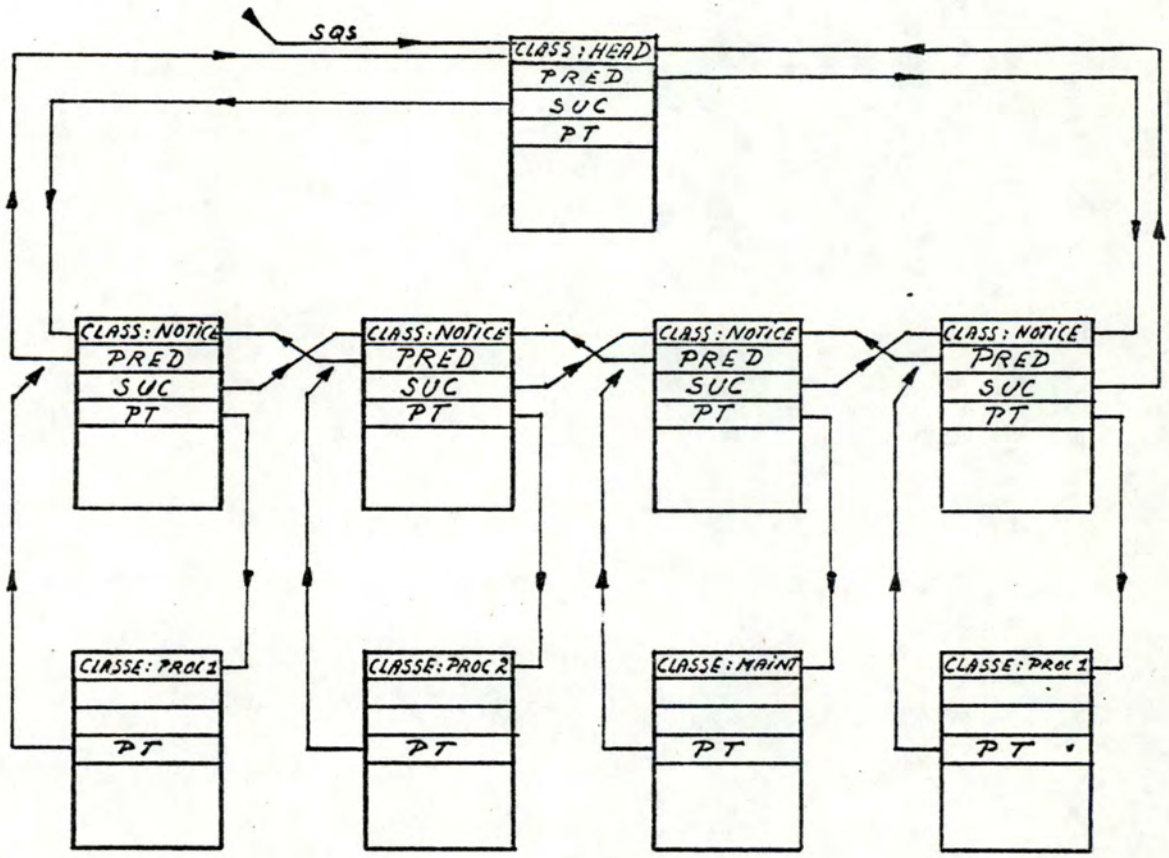


Figure 3-3: L'échéancier de SIMUFOR.

Signalons encore que l'on dénombre trois états possibles d'un processus, d'après l'événement éventuel qui lui est rattaché.

1. Un processus est dit "actif", ou "cédulé", s'il est rattaché à une notice d'événement qui se trouve chaînée dans la SQS. Ceci signifie qu'il y a une phase active cédulée pour ce processus.

2. Le processus courant (il n'y en a toujours qu'un seul à un moment précis de la simulation) est celui dont la notice d'événement se trouve en tête de l'échéancier. Il est pointé par la variable globale ACTUEL. Il représente un cas particulier de processus "actif".

3. Enfin, un processus est dit "passif", ou "idle" (du terme anglais de SIMULA), lorsqu'aucun événement le concernant n'est cédulé à ce moment.

La fonction booléenne IDLE permet de tester l'état "actif" ou "passif" d'un processus, à tout moment de la simulation.

3.7.12. La classe MAINT.

Nous aimerions pouvoir garder, tout au long de la simulation, le processus courant en tête de l'échéancier. Il suffit pour cela de ne pas enlever la notice d'événement de la SQS lors de la prise en charge d'un événement mais bien lors de la fin de la phase active qu'il a déclenchée (avant l'appel à RESUME qui prend en charge l'événement suivant). C'est possible pour tous les processus de la simulation. Cependant, en y réfléchissant bien, on découvre un problème.

Le programme principal de simulation n'est pas un processus, au sens de SIMUFOR. Pourtant, comme en SIMULA, nous aimerions pouvoir écrire dans le programme principal l'ordre HOLD (delay). Or, l'appel à HOLD implique, implicitement, que le programme appelant soit un processus. Nous avons résolu ce problème en créant une classe d'objets particulière: la classe MAINT qui porte le numéro 4. Celle-ci n'est pas accessible à l'utilisateur. De plus, un seul objet de cette classe est créé dans le sous-programme INIT. Il représente le programme principal, tout en faisant de lui un processus (principal). En effet, cet objet possédera:

- un numéro de classe;
- deux pointeurs (PRED et SUC) permettant de le mettre dans une liste;

- un pointeur de réactivation (LSC) pour sauver l'adresse du début de la prochaine phase active du processus;
- un pointeur (PT) permettant le lien avec une notice d'événement.

Ces variables-système permettent au programme principal de se comporter comme n'importe quel autre processus.

Les variables privées du processus principal sont uniques, car il n'existe jamais qu'un processus de ce type. Dès lors, ces variables ne devront pas être implantées en "zone dynamique", ni être adressées par la méthode des tableaux fictifs.

Dès que l'objet de la classe MAINT et la SOS auront été créés dans INIT, on créera une notice d'événement avec un temps nul que l'on associera au processus principal et, on l'insérera dans l'échéancier. Si bien qu'à la sortie du sous-programme INIT, appelé obligatoirement au début de toute simulation, on aura la situation suivante:

voir figure 3-4.

3.7.13. La priorité des processus.

Les processus possèdent une variable entière qui détermine leur priorité. C'est le sixième attribut-système des objets-processus (on le dénomme PRIOR). Lors de la création d'un processus par la fonction NEW, cette variable est initialisée à la valeur nulle. Cependant, la priorité d'un processus peut être modifiée à tout instant par l'intermédiaire de la procédure SETPR.

Cette priorité sert, principalement, lors de l'insertion du processus en file d'attente devant un serveur (procédure PRINTO appelée par ENTST et ENTFAC) (cfr. L'extension au système GPSSS).

Cependant, on peut en tenir compte également dans la gestion des processus. Si l'utilisateur se sert des procédures PACTIV et PREACT à la place de ACTIV et REACT respectivement, la reprise du processus concerné sera cédulée à l'heure

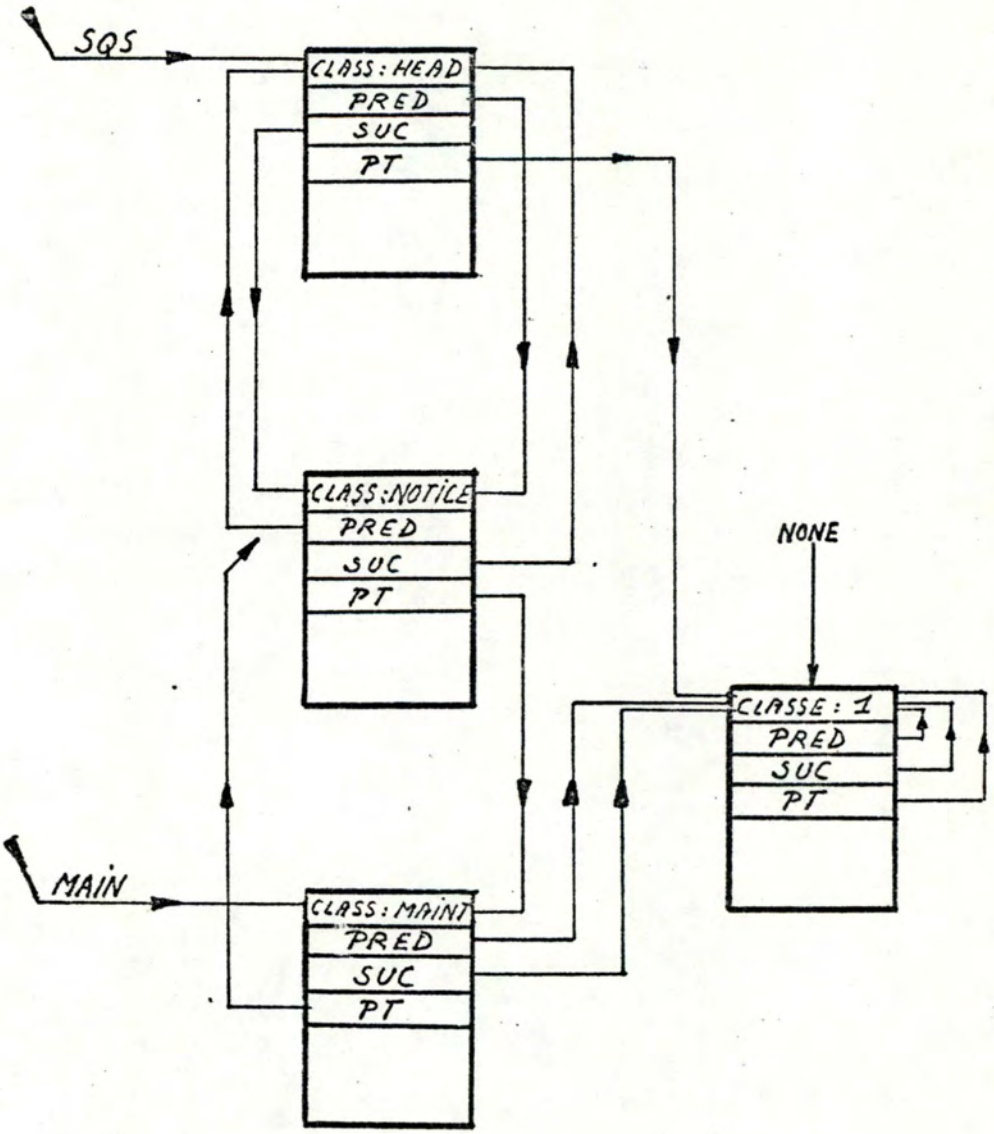


Figure 3-4: L'initialisation de l'échéancier.

prévue, mais avant tous les événements cédulés (à la même heure) pour un processus de priorité strictement inférieure. Par défaut, ACTIV et REACT place l'événement après tous ceux qui sont cédulés pour la même heure.

3.8. L'EXTENSION AU SYSTEME GPSSS.

3.8.1. Introduction.

Nous avons déjà indiqué la difficulté de programmer en SIMULA (et donc en SIMUFOR) des problèmes de simulation de réseaux de files d'attente. Nous avons donc tenté d'ajouter au noyau de SIMUFOR des procédures équivalentes à celles de GPSSS de J. G. VAUCHER [18], elles-mêmes inspirées du langage GPSS (cfr. chap. 2).

Nous examinerons successivement les différentes notions issues de GPSSS auxquelles nous nous sommes intéressés, et la façon dont nous les avons implémentées.

Notons tout d'abord que, contrairement à GPSSS qui se programme très proprement en SIMULA pur, nous avons été obligés de modifier certaines procédures de SIMUFOR pour pouvoir les utiliser dans la programmation des sous-programmes et des fonctions découlant de GPSSS. Nous pensons ici surtout aux sous-programmes de traitement de listes qui ont dû tenir compte du fait qu'une station simple et une station multiple, par exemple, sont assimilées à des têtes de liste. Ceci posait quelques problèmes en ce qui concerne certains contrôles de validité.

3.8.2. Les transactions.

Les principales entités introduites par GPSSS sont des entités passives. La seule entité active est la transaction et n'est rien d'autre qu'un processus légèrement étendu. Le terme transaction évoque cependant mieux le concept auquel nous avons affaire, à savoir le passage, pour une entité active (et mobile), d'une entité passive (et fixe) à une autre.

Nous avons déjà indiqué que tous les objets SIMUFOR (des classes créées par l'utilisateur) étaient considérés comme des processus. En fait, tout objet

possède les attributs d'un processus, bien que ces attributs ne soient pas toujours tous exploités.

Nous avons constaté que, pour faire de chaque processus SIMUFOR une transaction potentielle, il nous suffisait d'ajouter, à chaque objet, un dernier attribut-système (le septième, i.e. PTSTAT) qui ne sert qu'à mémoriser l'heure de passage de la transaction à certains endroits, de façon à pouvoir calculer des statistiques automatiquement.

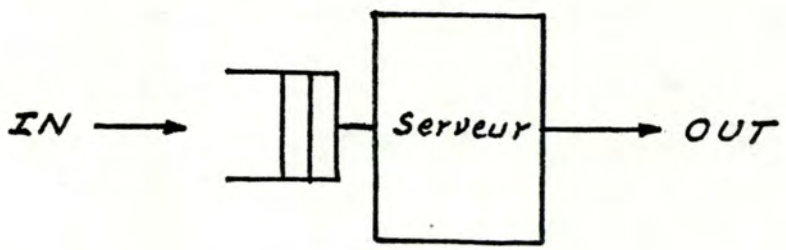
Avec ces sept attributs-système standards, chaque objet créé, d'une classe déclarée par l'utilisateur, est non seulement un processus potentiel mais aussi une transaction potentielle. Il ne faut rien déclarer ni ajouter à un processus quelconque pour lui faire jouer le rôle de transaction.

Par la suite, nous parlerons de transactions, pour garder l'esprit de GPSSS, tout en sachant bien qu'il s'agit de simples processus.

3.8.3. Les stations simples (ou "facilities").

La station simple est le premier type d'entités passives induites de GPSSS. On peut comparer une station simple à un guichet quelconque ou à une caisse de supermarché.

Son modèle est le suivant:



Elle est composée d'une file d'attente et d'un serveur unique.

Examinons le fonctionnement de cette entité. Lorsqu'une transaction demande à entrer dans la station, deux cas peuvent se produire:

1. le serveur est occupé, auquel cas la transaction se met en file d'attente;
2. le serveur est libre, auquel cas la transaction est servie[1] sans devoir attendre.

Lorsque le serveur a terminé de servir la transaction courante, il examine la file d'attente. Si celle-ci est vide, le serveur se "passive". Sinon, il prend en charge la transaction qui se trouve en tête de la file d'attente.

3.8.4. Les stations multiples (ou storages).

La station multiple se distingue de la station simple en ce sens que le serveur peut être considéré comme multiple et non plus unique.

La station multiple se compare à un parking d'un certain nombre de places, ou bien à un espace de mémoire d'un certain nombre de cadres. On peut considérer le serveur comme fractionné en un certain nombre d'unités. Ce nombre est appelé la capacité du storage. Une transaction demande, pour être servie, un certain nombre d'unités qu'elle libère à la sortie de la station. Par exemple, un programme de dix pages a besoin de dix cadres libres pour pouvoir être chargé en mémoire.

La file d'attente devant le serveur reste unique. Le schéma de la station multiple est donc le même que celui de la station simple avec, cependant, le fractionnement du serveur en plus.

Examinons le fonctionnement de cette entité. Lorsqu'une transaction demande à entrer dans la station, elle réclame un certain nombre d'unités (D)

---

[1] Remarquons que, ici comme par la suite, le fait pour une transaction de se faire servir par un serveur correspond à la poursuite de l'exécution du processus (i.e. de la transaction).

pour être servie. Deux cas peuvent se produire:

1. le nombre d'unités libres du serveur (NL) est supérieur au nombre d'unités demandées (D). Dans ce cas, le nombre d'unités libres est décrémenté de D (i.e. NL = NL - D) et la transaction est servie immédiatement;

2. le nombre d'unités libres du serveur est strictement inférieur au nombre d'unités demandées. Dans ce cas, la transaction se place en file d'attente.

Lorsque le serveur a terminé de s'occuper d'une transaction, il parcourt la file d'attente pour prendre en charge un nombre maximum de nouvelles transactions, tout en restant dans les limites de sa capacité (en termes d'unités disponibles).

3.8.5. Les groupes.

Un groupe est une entité passive qui se distingue assez fortement des stations simples et multiples.

On peut comparer un groupe à une benne de téléphérique qui ne se met en mouvement que lorsqu'un certain quota de personnes est atteint.

Lorsqu'une transaction arrive dans un groupe (ordre JOIN), son exécution s'arrête. Elle se met dans une liste des transactions bloquées dans le groupe. Supposons que l'on ait affaire à un groupe de dix unités. C'est seulement la dixième transaction qui libère les neuf précédentes. Elle-même passera sans attente. Les neuf suivantes seront à nouveau bloquées, et ainsi de suite...

3.8.6. Les régions.

Une région n'a aucun effet sur le déroulement des transactions qui y entrent et qui en sortent. Aucune file d'attente ne lui est donc associée. Cette entité a une utilité purement statistique. Elle permet de récolter des statistiques sur le temps de passage des transactions entre tel et tel point de

leur code.

On peut la comparer, par exemple, à un morceau d'autoroute situé entre deux sorties consécutives. Cependant, l'autoroute ne doit pas être encombrée, pour respecter la non-influence de la région sur le passage des transactions-voitures!

3.8.7. Les histogrammes.

Un histogramme reçoit, au cours de la simulation, un échantillon de valeurs. Il permet de visualiser la distribution de ces valeurs. L'affichage des histogrammes reste assez primitif (cfr. chap. 4). On pourrait envisager, par la suite, de le rendre plus agréable à l'oeil.

3.8.8. Quelques éléments d'implémentation.

Toutes les files d'attente dont il a été question ici seront, bien évidemment, implémentées sous la forme de doubles listes chaînées. Nous diposons déjà d'un ensemble de procédures de traitement adaptées à cette implémentation particulière de la notion de liste (qui a été choisie pour SIMUFOR).

Le chaînage des transactions dans les diverses files d'attente se fera à l'aide des deux attributs-système PRED et SUC. Il faut en tenir compte pour veiller à respecter la règle générale qui indique qu'un objet ne peut se trouver dans plus d'une liste à la fois (vue l'unicité du couple (pred, suc) pour un objet donné).

Notons qu'il y a une file d'attente par station simple, station multiple, ou groupe. Cette constatation nous a donné l'idée d'utiliser les stations elles-mêmes, et leur couple de pointeurs (PRED, SUC), comme tête de liste des files d'attente. De cette façon, on évite la création d'un tête de liste séparée par station. C'est la raison pour laquelle nous avons dû modifier

certaines procédures de traitement de listes. En effet, principalement lors du contôle de la validité de l'appel, la procédure INTO, par exemple, doit identifier sémantiquement les stations simples et multiples, ainsi que les groupes, à des têtes de liste.

D'autre part, indiquons que les classes prédéfinies numéro 5, 6, 7, 8 et 9 correspondent respectivement aux storages, facilités, groupes, régions et histogrammes.

3.8.9. La priorité des transactions et les files d'attente.

Examinons un instant la gestion de la file d'attente devant un serveur simple ou un serveur multiple (cas des facilités et des storages).

La discipline d'attente n'est pas FIFO (i.e. premier arrivé, premier servi), comme c'est le cas habituellement. La valeur de l'attribut-système de priorité (PRIOR) est pris en compte lorsqu'il faut placer la transaction en file d'attente. Les transactions prioritaires sont celles dont la priorité est la plus forte. A priorité égale et positive, la discipline d'attente reste FIFO. A priorité égale mais négative, elle devient LIFO (dernier arrivé, premier servi).

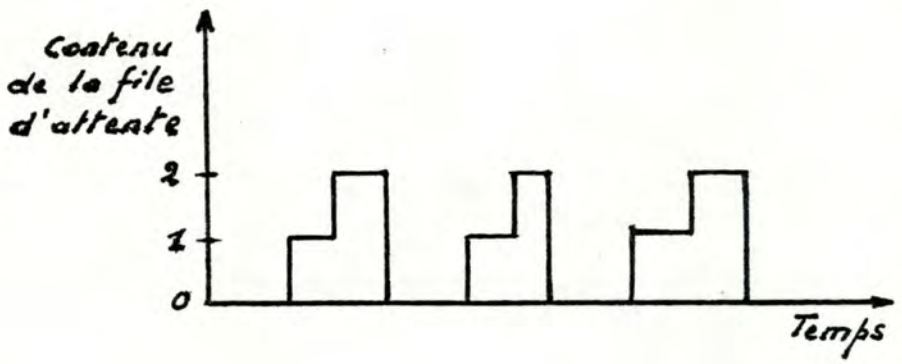
La priorité d'une transaction peut être modifiée par l'appel à la procédure SETPR. Si l'on ne fait pas appel à cette procédure (et donc par défaut), chaque transaction est créée avec une priorité nulle. Ceci assure une politique d'attente FIFO, par défaut, pour un ensemble de simulations qui reste, avouons-le, majoritaire.

3.8.10. Les statistiques.

La règle générale est de laisser l'initiative du calcul des statistiques à l'utilisateur. En effet, chaque cas de simulation demande un type de statistiques particulier. Aucun outil vraiment général ne peut être proposé.

Cependant, dans le cas des entités passives induites de GPSSS, certaines statistiques générales peuvent être proposées.

En ce qui concerne les groupes, nous ne calculerons aucune statistique. En effet celles-ci ne sauraient porter que sur l'attente des transactions. Or, elles seraient, en tout état de cause, biaisées. En effet, pour un groupe de trois éléments, par exemple, l'évolution de la file d'attente aura toujours la forme suivante:



Il reste à examiner les statistiques que nous calculerons pour les stations (simples et multiples) et les régions.

Nous avons a priori mis de côté les statistiques sophistiquées. Les tests de stationnarité des phénomènes d'attente ainsi que le calcul d'intervalles de confiance sur certaines variables[2] constituent certes des renseignements très intéressants, mais leur obtention nécessite la mémorisation de très gros volumes de données sur disque. D'autre part, l'effort d'analyse et de programmation (spécifique à ces problèmes statistiques complexes) pour arriver à de telles informations est appréciable. Nous n'en avons présentement pas le temps. Une version ultérieure de SIMUFOR pourrait, éventuellement, incorporer ces possibilités.

Dans cette première version, nous nous sommes contentés des informations

---

[2]Temps d'attente et de service, ainsi que longueurs de files d'attente, par exemple.

"cumulatives". Nous croyons avoir réussi à récolter toutes les statistiques possibles, pour autant qu'elles ne nécessitent pas la mémorisation d'une masse importante d'informations. Prenons un exemple. La moyenne et la variance d'un échantillon peut se calculer, par des formules cumulatives, au fur et à mesure de l'arrivée des données. Trois variables:

- le nombre de données déjà prises en compte;
- la moyenne partielle de ces données;
- la moyenne partielle des carrés des données;

suffisent, là où un fichier des données de l'échantillon aurait été nécessaire, sans l'utilisation des formules cumulatives.

Pour le calcul des statistiques sur la longueur des files d'attente, nous utiliserons la moyenne ergodique (ou temporelle) [6] qui ramène les calculs à l'échelle du temps simulé. Ce type de moyenne nous paraît exprimer, avec le plus de pertinence, l'information que nous désirons reproduire. D'autre part, nous avons inventé un indice que nous avons appelé "écart-type ergodique". Celui-ci est décrit dans la partie statistique du chapitre 4. Sa seule prétention est de donner une idée vague et intuitive de la dispersion de l'information autour de la moyenne ergodique.

Pour le calcul des temps d'attente et de service, nous mémoriserons l'heure d'entrée dans la file d'attente, et l'heure du début du service d'une transaction, dans l'attribut-système PTSTAT.

En ce qui concerne les stations simples, les statistiques et informations suivantes seront observées et affichées.

1. Des informations d'ordre général:

   - le nombre de transactions entrées dans la station;
   - le nombre de transactions entrées sans attente;
   - le pourcentage des transactions entrées sans attente;
   - la durée du temps d'observation des statistiques;
   - la durée du temps d'utilisation de la station;

- le taux d'utilisation de la station.

2. Des statistiques sur le temps d'attente devant le serveur. Ces statistiques sont conditionnelles au fait qu'il y ait effectivement eu une attente (les passages sans attente ne sont pas pris en compte et ils ont donc un poids nul). Les informations affichées à ce sujet sont:

   - le nombre d'observations recueillies;
   - la durée de l'attente la plus courte;
   - la durée de l'attente la plus longue;
   - la moyenne du temps d'attente devant le serveur;
   - l'écart-type du temps d'attente.

3. Des statistiques sur la longueur de la file d'attente devant le serveur, durant toute la période d'observation:

   - la longueur de la file d'attente au début de la période d'observation des statistiques (i.e. à "l'initialisation");
   - la longueur mimimale de la file d'attente durant la période d'observation;
   - la longueur maximale de la file durant cette période;
   - la moyenne (ergodique) de la longueur de la file sur toute la durée de cette période;
   - l'écart-type (ergodique) de la longueur de la file sur toute cette durée;
   - la longueur de la file d'attente au moment de l'impression des statistiques (i.e. en général, à la fin de la simulation).

4. Des statistiques sur le temps de service d'une transaction par le serveur:

   - le nombre d'observations recueillies pour calculer ces statistiques;
   - la durée du service le plus rapide;
   - la durée du service le plus lent;
   - la durée moyenne du service;
   - l'ecart-type du temps de service.

Pour les stations multiples, les informations affichées sont les mêmes que pour les stations simples mais il faut leur ajouter ce qui suit.

1. Des statistiques sur le nombre "d'unités de serveur" demandées par une transaction, pour pouvoir être servie. A cet effet, on affiche:

   - le nombre d'observations qui ont servi au calcul de ces statistiques;
   - la demande la plus modeste;
   - la demande la plus importante;
   - le nombre moyen d'unités demandées;
   - l'écart-type du nombre d'unités demandées.

2. Des statistiques sur le nombre "d'unités de serveur" occupées au cours de la durée du temps de saisie des informations. Comme pour la longueur des files d'attente, nous emploierons une moyenne ergodique qui tient bien compte de l'évolution dans le temps de l'occupation de la station (multiple). nous obtiendrons:

   - la capacité de la station (pour mémoire; c'est une donnée fixe);
   - l'occupation de la station au début de la période de récolte des statistiques;
   - l'occupation minimale de la station au cours de la période d'observation des statistiques;
   - l'occupation maximale de la station durant cette même période;
   - l'occupation moyenne (ergodique) de la station durant cette période;
   - l'écart-type (ergodique) du nombre d'unités occupées de la station;
   - l'occupation de la station lors de l'impression des statistiques.

Enfin, en ce qui concerne les régions, on s'intéressera aux informations suivantes.

1. Des statistiques sur le temps de passage de chacune des transactions à travers la région. Plus précisément, ce sont les informations suivantes:

   - la durée de la période d'observation des statistiques;
   - le nombre de transactions qui sont entrées dans la région durant cette période;
   - le nombre de transactions qui sont sorties de la région durant cette période;

- la durée du passage de la transaction la plus rapide;
- la durée du passage de la transaction la plus lente;
- la durée moyenne du passage des transactions dans la région;
- l'écart-type du temps de passage des transactions.

2. Des statistiques sur le nombre des transactions présentes dans la région tout au long de la période d'observation. Ici, il s'agira, à nouveau, de calculer une moyenne ergodique. Nous observerons:

   - le peuplement de la région au début de l'observation des statistiques;
   - l'occupation minimale de la région au cours de la période d'observation;
   - l'occupation maximale de la région au cours de cette période;
   - le peuplement moyen (ergodique) de la région durant cette période;
   - l'écart-type (ergodique) du contenu de la région durant cette période;
   - le peuplement de la région lors de l'impression des statistiques.

Nous nous permettrons encore une remarque importante à propos des statistiques. Pour une station simple ou multiple ou pour une région, la période d'observation des statistiques débute à la création de l'entité. L'impression des résultats des observations ne signifie pas l'initialisation d'une nouvelle période d'observation. Cependant, on peut forcer, à un quelconque moment de la simulation, l'initialisation partielle ou globale des observations statistiques. Cette possibilité est particulièrement avantageuse pour supprimer la partie non stationnaire (le démarrage de la simulation) d'un phénomène d'attente.

3.8.11. Les insuffisances et les précautions à prendre.

Deux éléments importants de GPSSS n'ont pas été implémentés à ce stade du développement de SIMUFOR.

1) D'abord, il n'existe pas de possibilité de préemption sur une station

simple ou une station multiple. Une transaction qui arrive ne peut donc pas replacer en file d'attente la transaction courante, pour se saisir du serveur.

2) Ensuite, la procédure WAIT-UNTIL n'est, elle non plus, pas encore implémentée. Cette procédure permet à une transaction d'arrêter son exécution jusqu'à ce qu'une condition donnée soit remplie. Elle est assez difficile à réaliser. En effet, il est difficile à un programme de déterminer quand la condition à remplir sera réalisée.

Notons enfin un point très important auquel l'utilisateur doit prendre garde. Le temps de passage intermédiaire d'une transaction à un endroit d'une station (simple ou multiple) ou d'une région est mémorisé dans un seul attribut-système (PTSTAT). Des statistiques se calculent par différence entre le temps simulé courant et le temps de passage mémorisé. On comprend, dès lors, l'incohérence de ces statistiques, qui découlerait du recouvrement du passage d'une transaction dans plusieurs stations ou régions. Un temps de passage mémorisé serait écrasé par une nouvelle valeur, avant même d'avoir pu être exploité.

Dès lors, supposons qu'une transaction soit entrée dans une station simple, par un ordre ENTFAC. Avant d'entrer dans une autre station simple, une station multiple, ou une région, cette transaction doit exécuter l'ordre LEAFAC pour sortir de la station où elle se trouve. Et il en est de même des stations multiples et des régions.

Donc, PAS DE RECOUVREMENTS en ce qui concerne l'usage des stations (simples ou multiples) et des régions.

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Institut d'Informatique

ANNÉE ACADÉMIQUE 1980-1981

UN SYSTÈME D'AIDE À LA SIMULATION À ÉVÉNEMENTS DISCRETS EN FORTRAN

2e partie

Jean-Mathieu NISEN
et
José ROULIN

Mémoire présenté
en vue de l'obtention du grade de
Licencié et Maître
en Informatique

TABLE DES MATIERES.

PREMIERE PARTIE.

DEUXIEME PARTIE.

TABLE DES FIGURES.

CHAPITRE 4

DESCRIPTION DES FONCTIONS ET DES SOUS-PROGAMMES

4. DESCRIPTION DES FONCTIONS ET DES SOUS-PROGRAMMES.

Dans ce chapitre, nous présentons une description aussi complète que possible de toutes les fonctions et de tous les sous-programmes qui constituent le code de SIMUFOR. Ceux-ci sont placés dans un ordre que nous pensons être logique et qui, par conséquent, ne nous paraît pas devoir être explicité davantage.

Pour toutes les fonctions et tous les sous-programmes, nous donnons lorsqu'il y a lieu:

1. la liste des arguments d'appel,
2. la description fonctionnelle,
3. la position dans la chaîne des appels,
4. et enfin, la description détaillée.

L'ensemble de ce chapitre est certainement de lecture assez pénible vu le nombre élevé de sous-programmes et fonctions présentés: nous avons, en effet, tenté de créer non seulement les outils nécessaires à une simulation mais aussi ceux qui nous ont paru susceptibles de simplifier l'usage de SIMUFOR.

Le Lecteur qui souhaite se limiter à la seule utilisation de SIMUFOR et ne désire pas le comprendre dans ses détails ou l'approfondir davantage, peut se borner à ne lire que les deux premiers des points cités ci-dessus: arguments d'appel et description fonctionnelle.

De son côté, le Lecteur qui voudrait faire une étude approfondie du présent chapitre gagnerait, selon nous, à examiner simultanément les pages qui suivent et le "listing" donné en Annexe.

Enfin, il va de soi qu'il est bon de se reporter, tout au long de l'étude de ce chapitre, aux paragraphes correspondants de la table des matières qui, mieux que nous pourrions le faire en phrases, donnent la succession des sous-

programmes et fonctions présentés ainsi que la façon dont ils sont groupés (sous-programmes et fonctions d'ordre général, gestion des processus, traitement de listes, ...). Nous ne reproduisons donc pas toute cette énumération ici.

4.1. SOUS-PROGRAMMES ET FONCTIONS D'ORDRE GENERAL.

4.1.1. Sous-programme INIT.

1) Arguments: nihil.

2) Description fonctionnelle:

Ce sous-programme réalise l'ensemble des initialisations nécessaires à un bon fonctionnement du système SIMUFOR. L'appel "call INIT" doit constituer la première instruction FORTRAN exécutable du programme principal de la simulation.

3) Position dans la chaîne des appels:

1. INIT appelle les sous-programmes INTO et LIEN, la fonction assembleur LOCF, ainsi que la fonction NEW.
2. Il doit être appelé par le programme principal.

4) Description détaillée:

- Plaçons-nous dans l'hypothèse où l'implantation des objets de la simulation se fait dans un grand commun blanc. Nous réservons la place en mémoire pour ce commun, par l'ordre COMMON ZAD (50000).
- Ceci nous donne une zone libre contiguë de 50000 (NAVAIL) mots-mémoire disponibles pour implanter les objets. Remarquons que l'on peut augmenter la dimension du commun blanc, pour autant que la taille du programme relié ne dépasse pas la taille de l'espace d'adressage disponible.
- LOCF (ZAD) nous fournit l'adresse du premier élément du commun blanc; tandis que LOCF (CLASS) nous fournit l'adresse d'implantation du premier tableau fictif du commun SYSVAR.
- Rappelons que tous les objets sont repérés, par la méthode des tableaux fictifs, à partir du commun SYSVAR. A ce point de la simulation, OBJLIB représente l'indice permettant d'atteindre le premier objet créé. Tout au long de la simulation, OBJLIB représentera toujours l'indice d'accès au prochain objet créé. MAXOBJ représente l'indice d'un objet hypothétique d'un seul attribut, implanté à l'extrémité du commun blanc. Cette notion permet de comparer les valeurs respectives de OBJLIB et de MAXOBJ pour constater directement la taille de la zone libre restant, à un moment donné, dans le commun blanc. OBJLIB et MAXOBJ représentent simplement un déplacement par rapport à LOCF(CLASS) plutôt qu'un déplacement par rapport à "zéro" (c'est-à-dire une adresse "absolue") ou par rapport à LOCF(ZAD) (c'est-à-dire le numéro d'un élément du tableau ZAD).
- Les attributs des vingt classes de SIMUFOR se trouvent dans trois vecteurs à vingt positions: les vecteurs SIZE, OLD et LSCO. Pour

$1 \leq i \leq 20$, le quadruplet (i, SIZE(i), OLD(i), LSCO(i)) représente le descripteur d'une classe SIMUFOR. Les vingt descripteurs de classes sont initialisés dans ce sous-programme INIT.

* Pour $1 \leq i \leq 20$, OLD(i) = NONE indique que la liste des objets détruits de chacune des vingt classes est vide (on n'a, en effet, encore créé aucun objet).

* Pour $1 \leq i \leq 20$, SIZE(i) = 7 indique que chaque objet de chaque classe possédera sept attributs par défaut. La composante SIZE d'un descripteur de classe est automatiquement modifiée par l'appel à la fonction CLASS.

* Pour $1 \leq i \leq 20$, LSCO(i) = 0 est une initialisation sans autre intérêt que celui d'éviter des variables non initialisées. Le "0" n'est pas significatif.

- L'objet NONE est le premier objet créé de la simulation.

- On déclare HEAD, NOTICE, MAINT, STORAG, FACILY, GROUPE, REGION et HISTO comme étant la représentation symbolique des classes numéro 2, 3, 4, 5, 6, 7, 8 et 9, respectivement.

- Si les objets des classes HEAD, NOTICE, MAINT, GROUPE et REGION ne possèdent que sept attributs, il n'en est pas de même pour les classes FACILY, STORAG et HISTO dont il faut ajuster la composante SIZE du descripteur.

- Les stations simples (facilités) sont liées entre elles dans une simple liste chaînée (voir fig. 4-1).

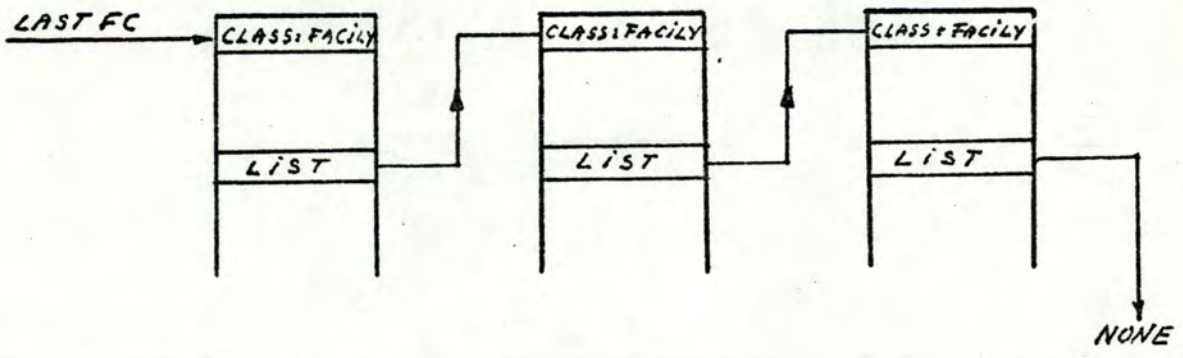


Figure 4-1: Chaînage des entités GPSSS.

- C'est de là que vient la nécessité, avant la création de la première station simple, d'initialiser LASTFC à NONE. Il en est de même pour les stations multiples , les groupes, les régions et les histogrammes. Ces chaînages permettent l'édition des statistiques de façon générique, par un seul appel à GENREP.

- Le temps simulé est initialisé à la valeur nulle (réelle) 0.0 .

- L'échéancier est créé, comme une simple liste, par l'appel à NEW (HEAD). La tête de liste de l'échéancier sera liée à un ensemble de notices d'événements. Lors d'un parcours de l'échéancier, sur le critère de l'heure des événements, il serait commode de pouvoir identifier la tête de liste comme un repère. C'est pourquoi nous initialiserons le pseudo-attribut EVTIME de la tête de liste de l'échéancier à la valeur -1.0. Cette heure ne peut être significative d'aucun événement. En effet, le temps simulé est initialisé à 0.0 et ne peut pas être décrémenté. En conséquence, il s'agit là d'un excellent point de repère.

- L'exécution de INIT se poursuit par la création d'un objet de la classe MAINT, pour faire du programme principal, un processus principal. Cet objet est repéré par le pointeur ACTUEL. En effet, au moment de l'exécution de INIT, le processus courant est bien le processus principal. Un notice d'événement est également créée avec une heure égale à 0.0 . Par une initialisation réciproque des pointeurs PT, elle est reliée au processus principal. Ensuite, elle est insérée dans l'échéancier. Elle représente d'ailleurs, à ce moment, la seule notice d'événement présente dans l'échéancier.

- La variable globale MAIN (du commun SYSVAR) pointera, durant toute la simulation, vers l'objet-processus principal. Ceci permet de programmer un ordre PASSIV dans le programme principal, puisqu'on garde un moyen de l'atteindre (pour l'activer) grâce à ce pointeur MAIN.

- L'initialisation des variables globales (du commun SYSVAR) AFTER, BEFORE, DELAY et AT permet l'emploi de noms symboliques dans l'appel des sous-programmes de cédulation des processus. L'effet recherché est de rendre les programmes de simulation plus parlants.

- L'initialisation de ITRACE à 0 signifie que par défaut, on ne trace pas l'exécution du programme.

- Par défaut, la lecture d'information se fait toujours au terminal. L'appel au sous-programme LIEN a pour but de demander à l'utilisateur où il désire imprimer les résultats de sa simulation (sur son terminal, à l'imprimante rapide ou bien sur disque, dans un fichier).

4.1.2. Fonction entière CLASS (loc, taille).

1) Arguments:

1. loc: contient l'adresse du début du code associé à la classe d'objets que l'on désire déclarer. Il s'agit de l'adresse du point d'entrée d'un sous-programme. On peut l'obtenir à l'aide de la fonction LOCF.

2. taille: contient le nombre d'attributs des objets de la classe. Le programmeur doit calculer cette taille de la façcon suivante: taille = 7 + N. Tout objet SIMUFOR possède en effet 7 attributs-système auxquels le programmeur ajoutera le nombre d'attributs (N), propre à la classe, qu'il juge nécessaire.

2) Description fonctionnelle:

- Cette fonction est réservée à l'utilisateur. Il doit l'appeler chaque fois qu'il désire créer une nouvelle classe d'objets de simulation.

- Cette fonction renvoie à l'utilisateur le numéro (cl) attribué par le système à la classe qu'il vient de créer. Ce numéro permettra de créer, simplement, des objets de cette classe par l'appel "objet = NEW (cl)", où "objet" représente un pointeur vers le nouvel objet créé.

3) Position dans la chaîne des appels:

1. ClASS appelle, dans certains cas, le sous-programme EERROR.

2. Elle peut être appelée par le programme principal et les sous-programmes de l'utilisateur.

4) Description détaillée:

- Par comparaison de FREECL et de MAXCL, on détermine s'il reste un descripteur de classe libre. Si ce n'est pas le cas, une erreur d'exécution est générée. Elle provoque l'impression d'un message d'erreur (numéro 101) et arrête la simulation.

- Si le nombre d'attributs (c'est-à-dire la taille) demandé pour les objets de la classe est inférieur à sept, une erreur d'exécution (affichage du message numéro 102 et arrêt la simulation) est également générée. En effet, la fonction NEW initialise par défaut les sept premiers attributs de chaque objet créé. Dès lors, si un objet possèdait moins de sept attributs, des initialisations seraient opérées à l'extérieur de l'espace de mémoire qui lui est réservé. Ceci rendrait, évidemment, la simulation incohérente.

- On donne à la classe créée le numéro du premier descripteur de classe libre (c'est-à-dire FREECL).

- On remplit ce descripteur avec les arguments de l'appel à la fonction CLASS:

    * la taille des objets de la classe est déterminée par l'argument TAILLE;

    * le pointeur vers la liste des objets détruits de la classe pointe vers l'objet NONE, indiquant qu'aucun objet de cette classe n'a encore été détruit (en effet, aucun objet de cette classe n'a encore été créé);

    * l'adresse du début du code associé à un objet-processus de cette nouvelle classe est donnée par l'argument LOC de l'appel à CLASS. Cependant, le démarrage de la première phase active d'un processus ne s'effectue pas par un simple branchement à l'adresse de début. On placera plutôt cette adresse au sommet du "stack global" (cfr. description des procédures ASSEMBLEUR) pour effectuer le branchement, classiquement chez DEC, par

l'instruction assembleur " PUSHJ 17,0". Dès lors, l'adresse du début du code associé sera sauvée dans le descripteur de classe sous la forme propre aux adresses sauvées dans la pile globale (chez DEC).

- Enfin, le numéro du premier descripteur de classe libre est incrémenté d'une unité.

4.1.3. Fonction entière NEW (klass).

1) Argument:

klass: contient le numéro de la classe dont on désire créer un nouveau représentant (un nouvel objet).

2) Description fonctionnelle:

- Cette fonction crée un nouvel objet de la classe dont le numéro est donné en argument.
- Elle alloue la place nécessaire à l'implantation de cet objet. Elle initialise l'ensemble des sept attributs-système standards de l'objet. Les attributs propres à cette classe particulière devront être initialisés par les soins de l'utilisateur, après l'appel à NEW.
- Enfin, elle renvoie à l'utilisateur l'indice permettant d'atteindre le nouvel objet créé.

3) Position dans la chaîne des appels:

1. NEW appelle, dans certains cas, le sous-programme EERROR.
2. Elle peut être appelée par le programme principal, les sous-programmes de l'utilisateur ainsi que par les sous-programmes ACTIV, REACT, PACTIV, PREACT et INIT.

4) Description détaillée:

- S'il existe un objet détruit de la classe concernée, la liste OLD (objets détruits de cette classe) n'est pas vide. On retire le premier objet de cette liste. C'est à la place qu'il occupe que l'on implantera alors le "nouvel" objet créé.
- Si la liste OLD est vide, le nouvel objet sera implanté à la suite de tous les autres objets déjà créés par la simulation, dans la zone "d'adressage dynamique", pour autant qu'il y reste suffisamment de place libre.

  * Si c'est le cas, la réservation de la place pour l'objet s'effectue en incrémentant simplement OBJLIB de la taille de l'objet.

* Si ce n'est pas le cas, une erreur d'exécution est générée. Il se produit donc l'affichage du message d'erreur numéro 103 et la simulation est arrêtée.

- Enfin, les attributs-système sont initialisés aux valeurs prévues. Rappelons que tout objet créé est considéré comme un processus.

  * Le numéro de classe (CLASS) correspond à la valeur de l'argument d'appel.

  * Les pointeurs PRED et SUC sont initialisés à NONE, indiquant que l'objet, à sa création, n'est chaîné dans aucune liste.
  Exception: si l'objet est une tête de liste (objet de la classe HEAD, numéro 2), les pointeurs PRED et SUC sont initialisés de manière à pointer chacun vers la tête de liste elle-même. Cette convention a été précisée au chap. 3.

  * Le pointeur PT est initialisé à NONE; ceci indique qu'aucun événement n'est cédulé, à sa création, pour le nouveau processus.

  * L'attribut LSC prend comme valeur l'adresse "formatée" du début du code associé au nouveau processus. Cette adresse est disponible dans le descripteur de la classe considérée.

  * La priorité du nouveau processus (et donc de la nouvelle transaction) est initialisée à la valeur nulle (valeur par défaut).

  * L'attribut-système PTSTAT n'a pas à être initialisé pour la cohérence de la simulation. Cependant, nous lui donnerons la valeur 0.0 dans le seul but de ne pas avoir de variables non initialisées.

4.1.4. Sous-programme système RETURN (obj).

1) Argument:

obj: contient l'indice d'un objet à détruire.

2) Description fonctionnnelle:

- Ce sous-programme détruit l'objet désigné par "obj". La destruction d'un objet de la simulation se résume à deux opérations importantes:

  1. D'une part, la place que l'objet occupait en mémoire est libérée. Ceci permet sa réutilisation éventuelle dans la suite de la simulation.

  2. D'autre part, tous les liens qui existaient entre cet objet et le reste des données de la simulation, sont brisés. Cette précaution s'impose afin d'éviter de confondre cet objet (qui n'existe virtuellement plus), avec un autre, implanté, par la suite, à la place du premier.

- Ce sous-programme constitue la charpente maîtresse de notre "collecteur de miettes" dont nous connaissons bien l'importance. Les procédures de gestion des processus l'utilisent pour détruire des notices d'événements devenues inutiles. Quant à l'utilisateur, il doit faire appel à ENDPRO pour désigner la fin d'un processus. C'est ENDPRO qui se charge d'appeler RETURN pour détruire le processus. Si l'utilisateur n'oublie pas de se servir d'ENDPRO, la récupération de place-mémoire s'effectuera, en très grande partie, automatiquement. Il ne lui restera éventuellement plus qu'à détruire des têtes de liste, des entités GPSSS, ou bien certains objets non utilisés en tant que processus et qui seraient devenus inutiles. Pour ce faire, il dispose du sous-programme KILL qui veille, lors de chaque destruction, à maintenir (dans la mesure du possible) la cohérence de la simulation.

3) Position dans la chaîne des appels:

1. RETURN appelle les sous-programmes EERROR et SOUT.

2. Il peut être appelé par les sous-programmes REACT, PREACT, PASSIV, CANCEL, WAIT, ENDPRO, ENTST, ENTFAC, KILL et JOIN.

4) Description détaillée:

- Une tentative de destruction de l'objet NONE se solde par l'affichage d'un message d'erreur (numéro 104) et par l'arrêt de la simulation.

- Si l'objet à détruire est membre d'une liste, il en est au préalable retiré.

- Ensuite, l'objet va rejoindre la liste des objets détruits de la classe dont il fait partie. Ceci rend la place qu'il occupait en mémoire, disponible pour l'implantation ultérieure d'un nouvel objet de la même classe que lui.

- Enfin, le pointeur passé comme paramètre actuel à RETURN est réinitialisé, afin de pointer vers l'objet NONE. Ceci permet de couper un lien qui existait entre l'objet détruit et les autres données de la simulation. Remarquons que seul le pointeur désigné par "obj" est ré-initialisé. S'il y en a d'autres pointant vers cet objet, ils ne sont pas réinitialisés. Ceci signifie que notre "collecteur de miettes" ne satisfait pas à l'entièreté des spécifications fonctionnelles demandées à un "garbage collector". Le respect de l'isolement total d'un objet détruit ne peut pas être obtenu avec les outils dont nous disposons. Seules quelques précautions supplémentaires pourront être prises dans le sous-programme KILL. Au programmeur à être conséquent et à se montrer prudent en ce qui concerne la destruction des objets.

4.1.5. Sous-programme KILL (obj).

1) Argument:

obj: contient l'indice d'un objet à détruire.

2) Description fonctionnelle:

- KILL assure la destruction de l'objet désigné par "obj". Il constitue une extension du sous-programme système RETURN, à l'intention de l'utilisateur. Certaines précautions complémentaires sont prises pour assurer la cohérence de la simulation, dans la mesure du possible, mais dans la limite de ce qui a été dit à la fin de la description détaillée de RETURN. Un collecteur de miettes tel que le nôtre ne pourra jamais atteindre la rigueur et l'efficacité d'un collecteur automatique. La réalisation de ce dernier demanderait, tout comme la recherche d'une meilleure organisation de l'échéancier, une vaste étude spécifique.

- Rappelons que, pour détruire un processus, l'utilisateur doit employer le sous-programme ENDPRO. L'utilisation de KILL pour détruire un objet-processus pourrait, dans certains cas, provoquer une incohérence de la simulation.

3) Position dans la chaîne des appels:

1. KILL appelle les sous-programmes RETURN, EERROR et WERROR.

2. Il peut être appelé par le programme principal ainsi que les sous-programmes de l'utilisateur.

4) Description détaillée:

- Une tentative de destruction de l'objet associé au programme principal se solde par l'affichage d'un message d'erreur (numéro 110) et l'arrêt de la simulation.

- Une tête de liste ne peut être détruite que si la liste correspondante est vide. Cette condition s'applique non seulement aux objets de la classe HEAD, mais aussi à ceux des classes STORAG, FACILY et GROUPE, qui peuvent être assimilés à des têtes de liste (file d'attente). Dans le cas d'une liste non vide, un message d'avertissement est affiché, mais la simulation se poursuit. N'oublions cependant pas que le cinquième avertissement signifie l'arrêt de la simulation. La solution consiste évidemment à tester le contenu d'une liste (à l'aide de la fonction EMPTY) avant de détruire la tête de liste correspondante.

- Lors de la destruction d'une entité déduite de GPSSS (storage, facilité, groupe, région ou histogramme), il faut veiller à maintenir le chaînage (entre elles, par leur attribut LIST) des entités "actives" de même type. Ceci revient à enlever de cette chaîne (simple liste chaînée), l'objet que l'on désire détruire. La

difficulté que nous avons éprouvée pour programmer ce retrait montre assez la supériorité des doubles listes chaînées, munies d'une tête de liste.

4.1.6. Sous-programme système LIEN.

1) Arguments: nihil.

2) Description fonctionnellle:

- Le sous-programme LIEN est appelé à la fin du sous-programme (INIT) d'initialisation du système SIMUFOR. Il permet à l'utilisateur de déterminer où il désire obtenir les résultats de sa simulation.
- Trois possibilités s'offrent à lui:

  * l'imprimante rapide du centre de calcul (OUTPUT = 3);
  * son terminal (OUTPUT = 5);
  * un fichier sur disque ($15 \leq$ OUTPUT ≤ 64).

3) Position dans la chaîne des appels:

LIEN est appelé par le sous-programme INIT.

4) Description détaillée:

- Toutes les instructions d'écriture de SIMUFOR sont libellées sous la forme: WRITE (OUTPUT, label).
- Nous conseillons à l'utilisateur de libeller ses ordres d'impression de la même manière (en n'oubliant pas d'inclure la déclaration du commun IO qui contient la valeur actuelle de la variable OUTPUT). De cette manière, tous les résultats seront imprimés à l'endroit déterminé lors de l'exécution de LIEN.

4.1.7. Sous-programme TEMPS (vartps).

1) Argument:

vartps: représente une variable dans laquelle on désire obtenir la valeur courante du temps simulé.

2) Description fonctionnelle:

Ce sous-programme renvoie simplement, dans le paramètre actuel de l'appel, la valeur de l'heure courante de la simulation.

L'utilisateur peut donc avoir accès à l'horloge de la simulation en lecture, mais non en écriture. Le temps simulé représente, en effet, une variable dont nous avons voulu protéger l'accès, en la déplaçant du commun SYSVAR (accessible à l'utilisateur) vers le commun TMPS (accessible seulement au système SIMUFOR). Il faut se rendre compte du fait qu'une modification intempestive de l'heure simulée peut rendre la simulation tout à fait incohérente.

3) Position dans la chaîne des appels:

TEMPS peut être appelé par le programme principal ainsi que par les sous-programmes de l'utilisateur.

4.1.8. Sous-programme SETPR (obj, val).

1) Arguments:

1. obj: contient l'indice d'un objet-processus dont on désire modifier la priorité.
2. val: contient la nouvelle valeur que l'on désire attribuer à la priorité du processus désigné par "obj".

2) Description fonctionnelle:

Le sous-programme SETPR permet à l'utilisateur de modifier la priorité du processus désigné par "obj", pour lui donner la nouvelle valeur "val".

3) Position dans la chaîne des appels:

SETPR peut être appelé par le programme principal ainsi que par les sous-programmes de l'utilisateur.

4) Description détaillée:

- Rappelons que la notion de priorité n'a de sens que pour un objet-processus (cf. chap. 3). Dès lors, on vérifie que "obj" désigne bien un processus, c'est-à-dire un objet d'une classe déclarée par l'utilisateur (il s'agit toujours d'un processus potentiel) ou bien l'objet pointé par MAIN (il représente le processus principal).
- Si l'objet désigné par "obj" est bien un processus, la nouvelle valeur de la priorité écrase l'ancienne. Si ce n'est pas le cas, aucune modification n'est apportée au sixième attribut-système de l'objet.

4.1.9. Fonction logique IDLE (proc).

1) Argument:

proc: contient l'indice d'un processus dont on désire connaître l'état.

2) Description fonctionnelle:

Cette fonction renvoie à l'utilisateur:

* la valeur VRAI si le processus est passif;

* la valeur FAUX si le processus est actif.

A un instant de la simulation, un processus est dit actif ou passif selon qu'il existe ou non (dans l'échéancier) un événement cédulé qui lui est rattaché.

3) Position dans la chaîne des appels:

1. IDLE appelle le sous-programme FERROR.
2. Elle peut être appelée par le programme principal ainsi que les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- Si "proc" ne désigne ni le processus principal, ni un processus de l'utilisateur, un message d'erreur (numéro 105) est affiché et la simulation s'arrête.
- La passivité d'un processus est acquise si aucune notice d'événement ne lui est rattachée (c'est-à-dire si PT (proc) = NONE).

4.1.10. Fonction réelle EVTIME (proc).

1) Argument:

proc: contient l'indice d'un processus.

2) Description fonctionnelle:

Cette fonction renvoie à l'utilisateur l'heure (simulée) de la prochaine phase active cédulée du processus désigné par "proc", pour autant que "proc" désigne bien un processus actif.

3) Position dans la chaîne des appels:

1. EVTIME appelle le sous-programme EERROR.

2. Elle peut être appelée par le programme principal ainsi que les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- Si "proc" ne désigne ni le processus principal, ni un processus de l'utilisateur, un message d'erreur (numéro 106) est affiché et la simulation s'arrête.

- Si le processus désigné par "proc" n'est pas actif (c'est-à-dire si aucune notice d'événement ne lui est rattachée) un autre message d'erreur (numéro 107) est affiché et la simulation s'arrête également.

- Si le processus désigné par "proc" est bien actif, l'heure cherchée est celle de l'attribut EVTIME de la notice d'événement qui lui est rattachée.

4.2. GESTION DES PROCESSUS.

4.2.1. Sous-programme système RESUME.

1) Arguments: nihil.

2) Description fonctionnelle:

Ce sous-programme gère l'évolution temporelle et la coordination des processus, et par conséquent la simulation elle-même: à chaque appel, il examine l'échéancier, définit le premier processus de celui-ci comme étant le processus courant, et met à l'heure l'horloge de la simulation.

3) Position dans la chaîne des appels:

1. RESUME appelle les sous-programmes DUMPS,SUIVRE, le sous-programme assembleur JUMPTO, et la fonction EMPTY.
2. Il est (ou peut être) appelé par les sous-programmes ENDPRO, ACTIV, REACT, PACTIV, PREACT, HOLD, PASSIV, CANCEL, WAIT.

4) Description détaillée:

- RESUME vérifie d'abord si la SQS est vide; dans ce cas, la simulation est arrêtée prématurément.
- Par contre, si elle n'est pas vide, RESUME définit le premier processus appartenant à la SQS comme processus courant. L'heure courante de la simulation (appelée TIME) est avancée à l'heure d'événement (EVTIME) de ce processus. Grâce à un appel à JUMPTO, le contrôle est passé au processus courant et l'exécution se déroule à partir de l'adresse de réactivation de ce processus; celle-ci avait été précédemment sauvegardée dans le pointeur de réactivation LSC de ce processus.

4.2.2. Sous-programme ACTIV (obj, code, dt).

1) Arguments:

1. obj: contient l'indice de l'objet à céduler;
2. code: indique le choix de la fonction (les valeurs possibles sont "at", "delay", "after", "before");
3. dt: contient, selon l'option choisie:

   - un délai (si le code est "delay");
   - une heure (si le code est "at");

- l'indice d'un autre objet-processus (si le code est "after" ou "before").

2) Description fonctionnelle:

A) activ obj [at / delay] dt

- L'objet désigné par "obj" doit être passif;
- "at" spécifie l'heure système de la phase active cédulée;
- "delay dt" est équivalent à "at TIME + dt" où TIME est l'heure courante du système;
- une notice d'événement correspondant à l'objet désigné par "obj" est insérée dans l'échéancier à l'heure système spécifiée, après chaque notice d'événement possédant la même heure système.
- Remarques:
  * "at dt" est équivalent à "at TIME" quand "dt" est inférieur à l'heure courante du système (TIME).
  * "delay dt" est équivalent à "delay 0.0" lorsque "dt" est négatif.
  * Dans les deux cas, un message d'avertissement est envoyé à l'utilisateur pour le prévenir que les paramètres qu'il a donnés sont erronés.

B) activ obj [after / before] dt

- Si "dt" est la référence à un objet-processus actif ou suspendu, la clause "before dt" ou "after dt" est utilisée pour insérer la notice d'événement de l'objet-processus designé par "obj" avant ou après celle de "dt" et à la même heure système.
- Si "dt" n'est ni un processus actif ni un processus suspendu, un message d'avertissement est transmis à l' utilisateur et l'appel du sous-programme n'a aucun autre effet.

3) Position dans la chaîne des appels:

1. ACTIV appelle le sous-programme assembleur SISAVE, les sous-programmes SFOLOW, SPRECE, EERROR, WERROR, RESUME, ainsi que la fonction NEW.

2. Il est (ou peut être) appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur, ainsi que les sous-programmes VERIST, LEAFAC, JOIN.

4) Description détaillée:

Remarques preliminaires:

* Nous avons déjà signalé (par. 3.2.1.) que la plupart des variables et constantes utilisées dans SIMUFOR sont entières (citons par exemple les indices qui permettent à l'utilisateur de faire référence aux objets qu'il crée). Néanmoins, nous devons parfois recourir à des variables réelles, en particulier pour représenter le temps simulé (variable TIME). D'après la description fonctionnelle du sous-programe ACTIV, il apparaît aisément qu'un des paramètres ("dt") a des modes différents selon l'option que l'on a choisie: ce paramètre est réel si le code est "at" ou "delay", il est entier si le code est "after" ou "before".

* Différentes solutions à ce problème existent, mais certaines sont contraignantes pour l'utilisateur; nous avons choisie celle qui consiste en l' emploi de deux variables supplémentaires "bidon" et "dtt" (la première réelle, la seconde entière), une équivalence entre "bidon" et "dtt" ainsi qu'une assignation "bidon=dtt". De cette manière, l'utilisateur n'est pas obligé de programmer, dans certains cas, la conversion du paramètre "dt".

* Notons enfin que, pour des raisons d'uniformité avec la description fonctionnelle, nous utiliserons toujours dans la suite de ce paragraphe, le symbole "dt" pour désigner le troisième argument du sous-programme; n'oublions cependant pas que, dans le code du sous-programme, ce symbole est remplacé par "dtt" lorsque le mode de la variable est entier !

Ce sous-programme ACTIV comprend cinq parties:

1. le sous-programme vérifie en premier lieu si l'objet à activer est NONE, auquel cas, un message d'avertissement est envoyé à l'utilisateur et l'appel de ce sous-programme n'a aucun autre effet.

2. Si l'objet à activer est différent de NONE, il y a sauvetage de l'adresse de réactivation dans la variable LSC conçue à cet effet.

3. Si l'objet à activer ne se trouve pas déjà dans la SQS, l'appel à la fonction NEW crée une notice d'événement. Par contre, si l'objet se trouve déjà dans la SQS, un message d'erreur est envoyé à l'utilisateur et la simulation est arrêtée prématurément.

4. Lorsque les deux premières vérifications conduisent à un résultat favorable, le sous-programme calcule l'endroit où il convient d'insérer la notice d'événement de l'objet à activer. Ce calcul diffère selon les différents codes ("delay", "at",

"after", "before"):

* Pour le code "delay":
  l'insertion dans la SQS est fonction de l'heure d'activation "t", somme de l'heure courante (TIME) et du délai "dt", passé comme paramètre. Lorsque "t" a été calculé, le parcours linéaire inversé (cf. remarque ci-après) de la SQS permet de trouver la notice d'événement (notée "i") dont l'heure d'événement est inférieure ou égale à "t". La notice d'événement de l'objet à activer est alors insérée dans la SQS directement après "i" à l'aide du sous-programme SFOLOW.

* Pour le code "at":
  l'insertion dans la SQS est, comme celle du code "delay", également fonction de l'heure d'activation "t" mais cette dernière est,dans ce cas, égale à l'heure "dt" passée comme paramètre. A nouveau, un parcours linéaire inversé permet de trouver l'endroit où a lieu l'insertion et celle-ci s'effectue à l'aide du sous-programme SFOLOW.

* Pour le code "after":
  ACTIV vérifie d'abord si l'objet désigné par "dt" (noté "objl") est actif ou suspendu.
  Si l'objet "objl" est bien actif ou suspendu, il existe dans la SQS une notice d'événement correspondant à cet objet (le pointeur PT qui relie l'objet "objl" et sa notice est différent de NONE); par un appel au sous-programme SFOLOW, la notice d'événement de l'objet à activer est alors insérée dans la SQS après la notice d'événement de l'objet "objl" et l'heure de sa phase active suivante est égale à l'heure d'événement de la notice du processus "objl".
  Par contre, si l'objet "objl" n'est ni actif, ni suspendu, un message d'avertissement est envoyé à l'utilisateur et l'appel à ACTIV n'a aucun autre effet.

* Pour le code "before":
  la procédure est identique à celle décrite pour le code "after", mais si besoin en est, l'insertion dans la SQS est effectuée grâce au sous-programme SPRECE.

* Remarque: le parcours de la SQS est parfois réalisé de façon inverse (c'est-à-dire en partant de la fin de la SQS et en progressant vers le début) car on peut, en effet, constater qu'en moyenne l'endroit où il convient d'insérer la notice d'événement est plus proche de la fin que du début de la SQS.

5. La notice d'événement qui vient d'être insérée dans la SQS est liée à l'objet "obj" par l'intermédiaire des deux pointeurs PT (celui de la notice et celui de l'objet "obj").

6. Comme pour tous les sous-programmes principaux de la gestion des processus, le contrôle est donné au sous-programme RESUME.

4.2.3. Sous-programme REACT (obj, code, dt).

1) Arguments:

1. obj: contient l'indice qui permet de référencer l'objet.
2. code: indique le choix de la fonction (les valeurs possibles sont "at", "delay", "after", "before").
3. dt: contient, suivant l'option choisie,

   - un délai (si le code est "delay"),
   - une heure (si le code est "at"),
   - l'indice d'un autre objet-processus (si le code est "after" ou "before").

2) Description fonctionnelle:

A) react obj [at / delay] dt

- L'objet désigné par "obj" peut être actif, passif ou suspendu.
- "at dt" spécifie l'heure système de la phase active cédulée.
- "delay dt" est équivalent à "at TIME + dt" où TIME est l'heure courante du système.
- La place, dans la SQS, de la notice d'événement de l'objet désigné par "obj", est modifiée de telle sorte qu'elle se trouve après chaque notice d'événement qui possède la même heure système.
- Remarques:

  * "at dt" est équivalent à "at TIME" quand "dt" est inférieur à l'heure courante du système (TIME);
  * "delay dt" , est équivalent à "delay 0.0" lorsque "dt" est négatif;
  * dans les deux cas, un message d'avertissement est envoyé à l'utilisateur.

B) react obj [after / before] dt

- Si "dt" est la référence à un objet-processus actif ou suspendu, la place, dans la SQS, de la notice d'événement de l'objet désigné par "obj" est modifiée de telle façon qu'elle se trouve avant (si le code est "before") ou après (si le code est "after") la notice de "dt" et à la même heure système que celle de cette dernière notice.

- Si "dt" n'est ni un processus actif ni un processus suspendu, un message d'avertissement est transmis à l' utilisateur et l'appel du sous-programme n'a aucun autre effet.

3) Position dans la chaîne des appels:

1. REACT appelle le sous-programme assembleur SISAVE, les sous-programmes SFOLOW, SPRECE, EERROR, WERROR, RESUME, RETURN, ainsi que la fonction NEW.

2. Il est (ou peut être) appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- De la description fonctionnelle des deux sous-programmes ACTIV et REACT, il apparaît clairement que l'un et l'autre doivent effectuer un traitement semblable, à la différence près que, dans le cas d'un appel au sous-programme ACTIV, le processus désigné par "obj" doit nécessairement être passif, alors que lors d'un appel à REACT, ce processus peut être actif, passif ou suspendu.

- Nous aurions souhaité ne pas allonger inutilement le code de SIMUFOR et réduire un appel à REACT en un appel à ACTIV. Malheureusement, cela ne pouvait être fait que par l'introduction de "drapeaux" ("flags") qui auraient non seulement ralenti l'exécution du programme mais aussi rendu malaisée la compréhension de ACTIV.

- Nous avons donc été obligés d'introduire le code en ligne de ACTIV, en y insérant évidemment la seule modification nécessaire: au lieu de générer un message d'erreur lorsque l'objet à activer se trouve déjà dans la SQS (cf. partie no. 3 de la description de ACTIV), REACT retire ce processus de la SQS, c'est-à-dire détruit sa notice d'événement par un appel à RETURN.

4.2.4. Sous-programme PACTIV (obj, code, dt).

1) Arguments:

1. obj: contient l'indice de l'objet à céduler;

2. code: indique le choix de la fonction (les valeurs possibles sont "at", "delay", "after", "before");

3. dt: contient, selon l'option choisie

   - un délai (si le code est "delay"),

- une heure (si le code est "at"),

- l'indice d'un autre objet-processus (si le code est "after" ou "before").

2) Description fonctionnelle:

pactiv obj [at / delay] dt

- L'objet désigné par "obj" doit être passif;

- "at dt" spécifie l'heure de la phase active cédulée;

- "delay dt" est équivalent à "at TIME + dt" où TIME est l'heure courante du système.

- Si

  * "proc" est l'objet-processus désigné par "obj",

  * "t" est l'heure système de la phase active cédulée suivante de "proc",

  * "p" est la priorité de "proc",

  une notice d'événement correspondant à "proc" est insérée dans la SQS au temps "t"

  * avant chaque notice d'événement qui possède la même heure système et dont la priorité du processus associé est inférieure à "p";

  * après chaque notice d'événement qui possède la même heure système mais dont la priorité du processus associé est supérieure ou égale à "p".

- Remarques:

  * "at dt" est équivalent à "at TIME" quand "dt" est inférieur à l'heure courante du système (TIME);

  * "delay dt" est équivalent à "delay 0.0" lorsque "dt" est négatif;

  * dans les deux cas, un message d'avertissement est envoyé à l'utilisateur.

3) Position dans la chaîne des appels:

1. PACTIV appelle les sous-programmes WERROR, FERROR, SPRECE, RESUME, le sous-programme assembleur SISAVE ainsi que la fonction NEW.

2. Il est (ou peut être) appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- PACTIV étant fort semblable au sous-programme ACTIV, nous n' en décrirons de façon détaillée que la partie concernant la priorité.
- Après avoir vérifié si l'objet-processus à activer n'est pas NONE, et si le code est valable (il n'y a aucun sens d' appeler ce sous-programme avec le code "after" ou "before"), PACTIV sauve l'adresse de réactivation du processus.
- Si le processus est actif ou suspendu, un message d'erreur qui entraîne l'arrêt de la simulation est envoyé à l'utilisateur.
- Par contre, si le processus est bien passif, l'appel à la fonction NEW créera une notice d'événement qui lui sera rattachée.
- Après avoir calculé l'heure d'activation "t" ("t" = "dt" si le code est "at", "t" = "dt + TIME" si le code est "delay"), un parcours linéaire de la SQS, tout en examinant les priorités des différents processus, permet de découvrir l'endroit, dans la SQS, où il convient d'insérer la notice d'événement de l'objet-processus désigné par "obj".
- Une fois l'insertion réalisée grâce à SPRECE, une fois le chaînage de la notice d'événement et de l'objet-processus effectué, le contrôle est donné au sous-programme RESUME.

4.2.5. Sous-programme PREACT(obj, code, dt).

1) Arguments:

1. obj: contient l'indice de l'objet à céduler;
2. code: indique le choix de la fonction (les valeurs possibles sont "at" et "delay");
3. dt: contient selon l'option choisie:
   - un délai (si le code est "delay");
   - une heure (si le code est "at").

2) Description fonctionnelle:

preact obj [at / delay] dt

- L'objet désigné par "obj" peut être actif, passif ou suspendu;

- "at dt" spécifie l'heure système de la phase active cédulée suivante;

- "delay dt" est équivalent à "at TIME + dt" où TIME est l'heure courante du système.

- Si

  * "proc" est le processus désigné par "obj",

  * "t" est l'heure de la phase active cédulée suivante de "proc",

  * "p" est la priorité de "proc",

  la place de la notice d'événement de "proc", dans la SQS, est modifiée de telle façon qu'elle se trouve à l'heure "t"

  * avant chaque notice d'événement qui possède la même heure système et dont la priorité du processus associé est inférieure à "p",

  * après chaque notice d'événement qui possède la même heure système mais dont la priorité du processus associé est supérieure ou égale à "p".

  Remarques:

  * "at dt" est équivalent à "at TIME" quand "dt" est inférieur à l'heure courante du système (TIME);

  * "delay dt" est équivalent à "delay 0.0" lorsque "dt" est négatif;

  * dans les deux cas, un message d'avertissement est envoyé à l'utilisateur.

3) Position dans la chaîne des appels:

1. PREACT appelle les sous-programmes RETURN, WERROR, SPRECE, RESUME, la fonction NEW ainsi que le sous-programme assembleur SISAVE.

2. Il est (ou peut être) appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

Comme nous avons dû le faire pour le sous-programme REACT (cf par. 4.2.3) nous devons introduire le code en ligne du sous-programme PACTIV dans PREACT en y apportant la même modification: destruction de la notice d'événement de l'objet-processus désigné par "obj" si celle-ci se trouve déjà dans la SQS.

4.2.6. Sous-programme HOLD (dt).

1) Argument:

dt: contient le délai.

2) Description fonctionnelle:

- HOLD recédule le processus courant de telle sorte que sa phase active suivante ait lieu à l'heure "TIME + dt" (TIME étant l'heure courante du système).
- Si la valeur du paramètre actuel "dt" est négative, l'appel est équivalent à HOLD (0.0).

3) Position dans la chaîne des appels:

1. HOLD appelle le sous-programme assembleur SISAVE, les sous-programmes WERROR, SFOLOW et RESUME.
2. Il peut être appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur ainsi que par le sous-programme JOIN.

4)Description détaillée:

- Après avoir sauvé l'adresse de réactivation dans la variable LSC conçue à cet effet, HOLD calcule et vérifie l'heure de la phase active cédulée suivante du processus courant.
- Un parcours linéaire inversé de la SQS permet de trouver l'endroit où devra être réinsérée la notice d'événement correspondant au processus courant.
- L'appel à SFOLOW modifiera la place de la notice d'événement du processus courant, tout en mettant à jour les différents pointeurs PRED et SUC de la notice d'événement associée à l'objet courant, ainsi qu'à celle des prédécesseurs et successeurs de cette notice avant et après la modification de place.
- Enfin, comme pour tous les sous-programmes de la gestion des processus, le contrôle est passé au sous-programme RESUME.

4.2.7. Sous-programme PASSIV.

1) Arguments : nihil.

2) Description fonctionnelle:

Un appel à ce sous-programme a pour effet de rendre passif

("passiver") le processus courant, c'est-à-dire de le retirer de la SQS.

3) Position dans la chaîne des appels:

1. PASSIV appelle le sous-programme assembleur SISAVE, ainsi que les sous-programmes RETURN et RESUME.
2. Il peut être appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- Après avoir sauvé l'adresse de réactivation du processus qui doit être passivé dans la variable LSC de celui-ci, PASSIV retire, de la SQS, (c'est-à-dire détruit) sa notice d'événement et met à jour les pointeurs PRED et SUC des notices d'événement de la SQS; ces deux opérations de destruction et de mise à jour sont réalisées par un appel au sous-programme RETURN.
- Enfin, comme pour tous les sous-programmes de la gestion des processus, le contrôle est donné à RESUME, qui sélectionne le processus suivant (celui-ci devient donc courant) et met à l'heure l'horloge de la simulation.

4.2.8. Sous-programme CANCEL (obj).

1) Argument:

obj: contient l'indice de l'objet-processus concerné.

2) Description fonctionnelle:

Un appel à CANCEL a pour effet de rendre passif le processus désigné par "obj", c'est-à-dire de le retirer de la SQS.

3) Position dans la chaîne des appels:

1. CANCEL appelle les sous-programmes RETURN, WERROR et RESUME ainsi que le sous-programme assembleur SISAVE.
2. Il peut être appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

CANCEL est fort semblable au sous-programme PASSIF, qui n'en est au fond qu'un cas particulier; c'est pourquoi, de la même manière, CANCEL

* sauve l'adresse de réactivation,

* vérifie ensuite si le processus à passiver possède une notice d'événement dans la SQS qu'il détruit par un appel à RETURN,

* met à jour les pointeurs PRED et SUC de la SQS,

* et enfin, génère un appel à RESUME.

4.2.9. Sous-programme WAIT (q).

1) Argument:

q: fournit l'indice qui permet de référencer une liste.

2) Description fonctionnelle:

Un appel à WAIT rend passif le processus courant, c'est-à-dire le retire de la SQS et le met à la fin de la file désignée par "q".

3) Position dans la chaîne des appels:

1. WAIT appelle le sous-programme assembleur SISAVE ainsi que les sous-programmes RETURN, INTO et RESUME.
2. Il peut être appelé par le programme principal, les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- WAIT sauve d'abord l'adresse de réactivation du processus courant.
- Ensuite, un appel à RETURN passive ce processus, c'est-à-dire le retire de la SQS, détruit sa notice d'événement et met à jour les pointeurs PRED et SUC de la SQS.
- Un appel à INTO chaîne ce processus passif à la fin de la file "q".
- Enfin, un appel à RESUME permet de sélectionner le nouveau processus courant.

4.2.10. Sous-programme ENDPRO.

1) Arguments: nihil.

2) Description fonctionnelle:

ENDPRO permet à l'utilisateur de détruire les objets devenus inactifs de sa simulation; cela permet ainsi de limiter le besoin global

d'espace mémoire.

3) Position dans la chaîne des appels:

1. ENDPRO appelle les sous-programmes RETURN et RESUME ainsi que le sous-programme assembleur NET.

2. Il doit être appelé à la fin de tous les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

Deux appels successifs au sous-programme RETURN permettent à ENDPRO de détruire l'objet-processus ainsi que sa notice d'événement si elle existe. Les appels à NET retirent, de la pile ("stack"), les adresses de retour devenues inutiles.

4.3. TRAITEMENT DE LISTES.

La notion de liste est essentielle en simulation. C'est pourquoi, nous avons dû l'implémenter dans SIMUFOR. Nous avons choisi la même implémentation que SIMULA: la double liste chaînée circulaire avec tête de liste. Rappelons qu'à cet effet, tous les objets créés par la simulation (sans exception) sont dotés des deux pointeurs: PRED et SUC. Ceux-ci permettent le chaînage de chaque objet dans une liste.

Une liste est construite de la manière suivante (voir fig. 4-2)

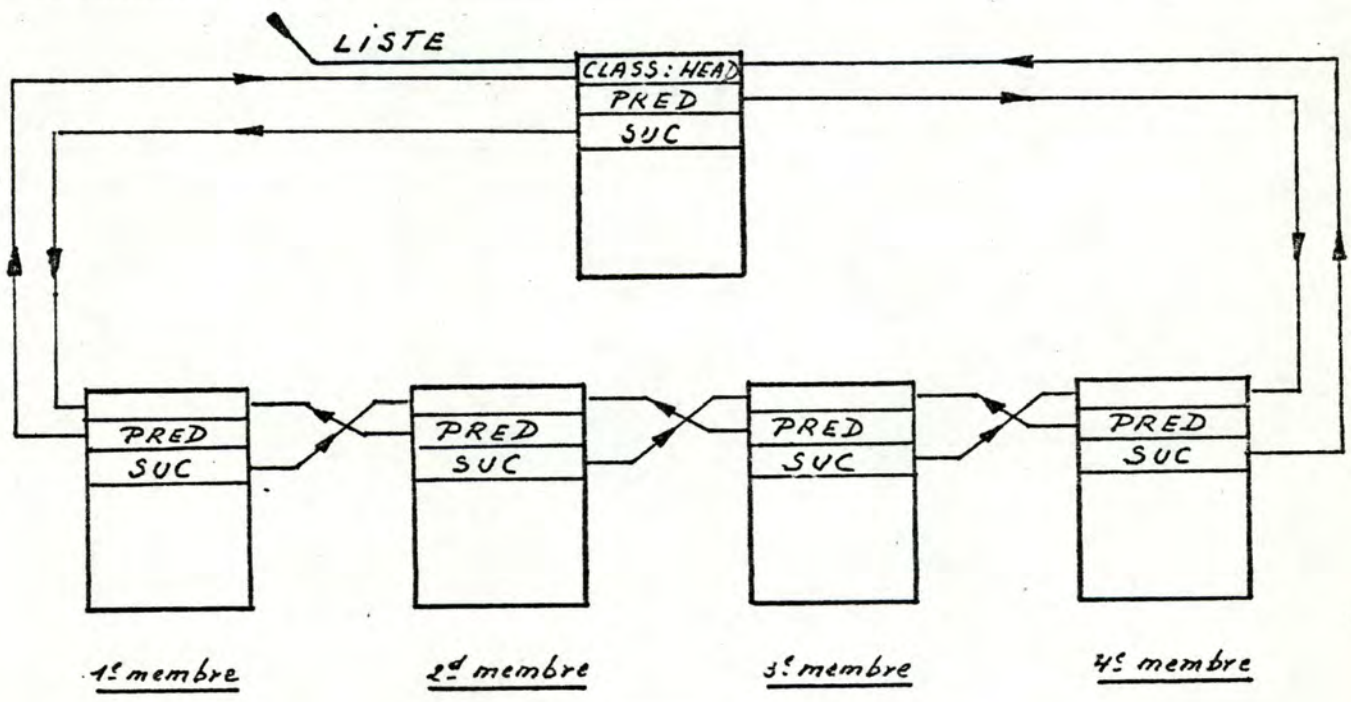


Figure 4-2: Structure d'une liste de SIMUFOR.

1. Elle possède une tête de liste (objet de la classe HEAD) dont les pointeurs SUC et PRED désignent respectivement le premier et le dernier membre de la liste. Par convention, si la liste est vide, les pointeurs SUC et PRED pointent tous les deux vers la tête de liste elle-même.

2. Si la liste n'est pas vide, elle possède un certain nombre de membres (c'est-à-dire un certain nombre d'objets qui en font partie). Chaque membre possède également les deux attributs SUC et PRED. Ici, SUC désigne toujours le membre suivant de la liste alors que PRED désigne toujours le membre précédent. Le pointeur PRED du premier membre ainsi que le pointeur SUC du dernier membre désignent la tête de liste.

Nous voyons donc qu'une liste est un ensemble chaîné et ordonné de membres.

Rappelons qu'un objet de SIMUFOR ne peut être membre que d'une liste à la fois. Par conséquent, à tout moment de la simulation, un objet peut:

- soit se trouver dans une (et une seule) liste;
- soit ne se trouver dans aucune liste, auquel cas ses attributs SUC et PRED pointent tous les deux vers l'objet NONE (ce qui signifie qu'ils ne pointent vers rien de significatif au sens de la simulation).

Nous allons le constater, l'implémentation choisie permet une programmation très simple des procédures de traitement de listes. Parmi ces procédures, trois sont réservées au système SIMUFOR. Il s'agit des sous-programmes SOUT, SPRECE et SFOLOW, d'ailleurs tous trois préfixés par la lettre S. Ils effectuent les opérations élémentaires de retrait et d'insertion. Les autres sont accessibles à l'utilisateur. Le traitement d'une liste et de ses membres se ramène presque toujours à la manipulation des deux seuls attributs-système PRED et SUC. Examinons à présent chaque procédure en détail.

4.3.1. Sous-programme système SOUT (j).

1) Argument:

j: contient l'indice de l'objet à extraire de la liste.

2) Description fonctionnelle:

- Le sous-programme SOUT enlève l'objet désigné par "obj" de la liste dans laquelle il se trouve.
- Il ne réalise aucune vérification de la cohérence de l'opération de retrait et ne réinitialise pas les pointeurs de l'objet extrait. Il est réservé au système SIMUFOR. L'utilisateur désireux de programmer la même opération doit utiliser le sous-programme OUT.

3) Position dans la chaîne des appels:

SOUT est appelé par les sous-programmes RETURN, SPRECE et OUT.

4) Description détaillée:

Le code de SOUT s'exécute en deux phases successives.

1. Il s'agit d'abord de repérer le successeur et le prédécesseur de l'objet désigné par "j".

2. Ensuite, on réalise la mise à jour du pointeur SUC du prédécesseur de "j" afin de le faire pointer vers le successeur de "j".

3. Enfin, on effectue la mise à jour du pointeur PRED du successeur de "j" afin de le faire pointer vers le prédécesseur de "j".

4.3.2. Sous-programme système SPRECE (j, k).

1) Arguments:

1. j: contient l'indice de l'objet à insérer dans la liste.

2. k: contient l'indice de l'objet avant lequel on désire insérer le précédent.

2) Description fonctionnelle:

- SPRECE insère l'objet désigné par "j" juste avant l'objet désigné par "k", à l'intérieur de la liste dont "k" est sensé faire partie.

- Si l'objet désigné par "j" appartient déjà à une liste, il en est au préalable retiré.

- Le sous-programme SPRECE ne réalise aucune vérification de la cohérence de l'opération d'insertion. Il est réservé au système SIMUFOR. L'utilisateur souhaitant programmer la même opération doit faire appel au sous-programme PRECED.

3) Position dans la chaîne des appels:

1. SPRECE appelle le sous-programme SOUT.

2. Il est appelé par les sous-programmes ACTIV, REACT, PACTIV, PREACT, SFOLOW, PRECED et PRINTO.

4) Description détaillée:

L'insertion s'opère en deux étapes successives:

1. Il s'agit d'abord d'identifier le prédécesseur de l'objet désigné par "k" afin de connaître les deux membres entre lesquels sera inséré l'objet désigné par "j". Soit "i" l'indice désignant le prédécesseur en question.

2. L'insertion proprement dite s'effectue par les quatre instructions suivantes:

```
PRED (j) = i
SUC  (j) = k
PRED (k) = j
SUC  (i) = j
```

Cette séquence d'instruction est suffisamment parlante pour ne pas devoir être commentée. Visualisons cependant l'insertion sur un dessin (cf . fig. 4-3).

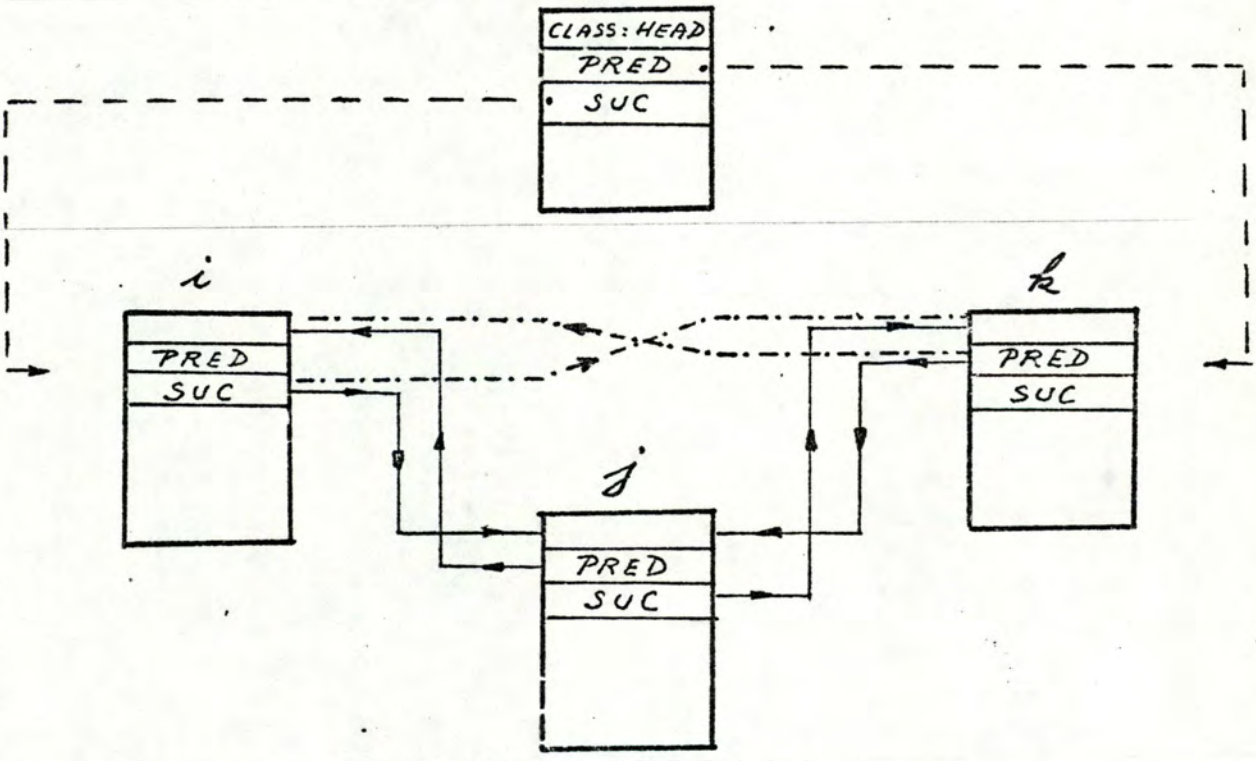


Figure 4-3: L'insertion d'un objet dans une liste.

4.3.3. Sous-programme système SFOLOW (j, i).

1) Arguments:

1. j: contient l'indice de l'objet à insérer dans la liste.
2. i: contient l'indice de l'objet à la suite duquel on désire insérer le précédent.

2) Description fonctionnelle:

- Le sous-programme SFOLOW insère l'objet désigné par "j" juste après l'objet désigné par "i", à l'intérieur de la liste dont "i" est sensé faire partie.
- Si l'objet "j" appartient déjà à une liste, il en est au préalable retiré.
- Tout comme SPRECE, le sous-programme SFOLOW ne réalise aucune vérification de la cohérence de l'opération d'insertion. Il est également réservé au système SIMUFOR. L'utilisateur souhaitant programmer la même opération doit faire appel au sous-programme FOLLOW.

3) Position dans la chaîne des appels:

1. SFOLOW appelle le sous-programme SPRECE.

2. Il est appelé par les sous-programmes ACTIV, REACT, HOLD et PRINTO.

4) Description détaillée:

Nous essayerons tout simplement de nous placer dans le contexte d'utilisation du sous-programme SPRECE. Il suffit à cet effet de faire la remarque suivante: l'opération d'insertion de l'objet désigné par "j" à la suite immédiate de celui désigné par "i" équivaut à l'insertion de ce même objet juste avant le successeur de "i".

4.3.4. Sous-programme OUT (i).

1) Argument:

i: contient l'indice de l'objet à retirer de la liste.

2) Description fonctionnelle:

- Tout comme SOUT, le sous-programme OUT enlève l'objet désigné par "i" de la liste dans laquelle il se trouve.
- Cependant, la cohérence de l'opéretion de retrait est assurée par une série de tests supplémentaires. Cette précaution rend la procédure OUT directement opérationnelle pour l'utilisateur de SIMUFOR.

3) Position dans la chaîne des appels:

1. OUT appelle les sous-programmes EERROR et SOUT.

2. Il est appelé par les sous-programmes LEAFAC, JOIN et VERIST.

4) Description détaillée:

- Avant d'appeler SOUT, nous prendrons deux précautions.

    1. Tout d'abord, on ne peut pas retirer la tête d'une liste. Si l'objet désigné par "i" est une tête de liste, un message d'erreur (numéro 301) est affiché et la simulation est interrompue.

    2. Ensuite, nous nous assurons que l'objet désigné par "i" appartient bien à une liste. Si ce n'est pas le cas, le contrôle est immédiatement rendu au programme appelant.

- Après l'appel à SOUT, les pointeurs PRED et SUC de l'objet "i" sont réinitialisés pour pointer vers l'objet NONE, ce qui indique que l'objet "i" n'appartient plus à une liste.

4.3.5. Sous-programme PRECED (j, k).

1) Arguments:

1. j: contient l'indice de l'objet à insérer dans la liste.
2. k: contient l'indice de l'objet avant lequel on désire insérer "j".

2) Description fonctionnelle:

- Le sous-programme PRECED insère l'objet désigné par "j" juste avant l'objet désigné par "k", à l'intérieur de la liste dont "k" est sensé faire partie.
- Si l'objet désigné par "j" appartient déjà à une liste, il en est au préalable retiré.
- Le sous-programme PRECED a été écrit à l'intention de l'utilisateur de SIMUFOR. Par rapport à SPRECED, il effectue en plus un test de cohérence de l'insertion demandée.

3) Position dans la chaîne des appels:

1. PRECED appelle les sous-programmes EERROR et SPRECE.
2. Il est appelé par les sous-programmes FOLLOW et INTO.

4) Description détaillée:

Avant d'appeler SPRECE (avec les mêmes arguments que PRECED), nous vérifierons que les conditions suivantes sont remplies:

* l'objet "i" que l'on désire insérer n'est ni l'objet NONE, ni une tête de liste;
* l'objet désigné par "k" n'est pas l'objet NONE;
* le successeur de l'objet désigné par "k" n'est pas, lui non plus, l'objet NONE.

Si l'une ou l'autre de ces trois conditions n'est pas remplie, un message d'erreur (numéro 302) est affiché et la simulation s'arrête. Remarquons que les deux dernières conditions signifient que l'objet désigné par "k" doit absolument faire partie d'une liste.

4.3.6. Sous-programme FOLLOW (j, i).

1) Arguments:

1. j: contient l'indice de l'objet à insérer dans la liste.
2. i: contient l'indice de l'objet à la suite duquel on désire insérer le précédent.

2) Description fonctionnelle:

- Le sous-programme FOLLOW insère l'objet désigné par "j" juste à la suite de l'objet désigné par "i", à l'intérieur de la liste dont "i" est sensé faire partie.
- Si l'objet désigné par "j" appartient déjà à une liste, il en est au préalable retiré.
- Tout comme PRECED, le sous-programme FOLLOW est destiné à l'utilisateur de SIMUFOR. Par rapport à SFOLOW, il effectue un test supplémentaire de cohérence de l'insertion demandée.

3) Position dans la chaîne des appels:

1. FOLLOW appelle le sous-programme PRECED.
2. Il peut être appelé par le programme principal ainsi que les sous-programmes de l'utilisateur.

4) Description détaillée:

- La stratégie utilisée est la même que celle dont nous nous sommes servis pour l'écriture de SFOLOW. Nous utiliserons simplement la procédure PRECED pour insérer l'objet désigné par "j" juste avant le successeur de "i" (ce qui équivaut à l'insérer juste après l'objet "i" lui-même).
- L'appel à PRECED provoque automatiquement la vérification des trois conditions définies dans la description détaillée de cette procédure. La cohérence de l'insertion est donc assurée.

4.3.7. Sous-programme INTO (i, l).

1) Arguments:

1. i: contient l'indice de l'objet à insérer à la fin de la liste.
2. l: contient l'indice d'une tête de liste.

2) Description fonctionnelle:

Le sous-programme INTO insère l'objet désigné par "i" à l'extrémité de la liste dont la tête est désignée par "l". Il est destiné à faciliter la gestion des listes particulières que sont les files d'attente. En effet, dans de nombreux phénomènes stochastiques, la politique d'attente est "FIFO". Dans ce cas, un nouvel arrivant se place toujours à la fin de la file d'attente, d'où l'intérêt de INTO.

3) Position dans la chaîne des appels:

1. INTO appelle les sous-programmes EERROR et PRECED.
2. Il peut être appelé par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que les sous-programmes WAIT, JOIN, INIT et PRINTO.

4) Description détaillée:

- L'insertion s'opère par l'appel CALL PRECED (i, l) qui insère l'objet juste avant la tête de liste (c'est-à-dire tout à la fin de la liste).
- Cependant, avant d'effectuer l'insertion, il faut s'assurer que "l" désigne un objet de la classe HEAD, ou bien un objet assimilable à une tête de liste (c'est-à-dire un objet des classes STORAG, FACILY ou GROUPE) (cfr. chap. 3: L'extention au système GPSSS). Si cette condition n'est pas vérifiée, il y aura affichage d'un message d'erreur (numéro 303) et arrêt de la simulation.

4.3.8. Fonction logique EMPTY (l).

1) Argument:

l: contient l'indice d'une tête de liste.

2) Description fonctionnelle:

La fonction EMPTY renvoie la valeur booléenne "VRAI" (.true. en FORTRAN) si la liste dont la tête est désignée par "l" est vide. Elle renvoie la valeur booléenne "FAUX" (.false. en FORTRAN) dans le cas contraire.

3) Position dans la chaîne des appels:

1. EMPTY appelle le sous-programme EERROR.
2. Elle peut être appelée par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que le sous-programme RESUME et les fonctions FIRST et LAST.

4) Description détaillée:

- Nous nous assurerons tout d'abord que nous avons bien affaire à une liste. Il nous faut donc vérifier que "1" désigne bien une tête de liste ou un objet assimilable à une tête de liste. Si ce n'est pas le cas, un message d'erreur (numéro 304) sera affiché et la simulation s'arrêtera.

- Pour déterminer si la liste est vide, nous utiliserons la convention que nous avons déjà souvent exprimée: les pointeurs PRED et SUC d'une tête de liste vide pointent, tous les deux, vers la tête de liste elle-même.

4.3.9. Fonction entière FIRST (1).

1) Argument:

1: contient l'indice d'une tête de liste.

2) Description fonctionnelle:

- La fonction FIRST renvoie à l'utilisateur l'indice du premier membre de la liste dont la tête est désignée par "1".

- Si la liste est vide, FIRST renvoie un pointeur vers l'objet NONE. Cela signifie évidemment qu'il n'existe pas de premier membre dans la liste.

3) Position dans la chaîne des appels:

1. FIRST appelle la fonction EMPTY.

2. Elle peut être appelée par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que les sous-programmes LEAFAC, VERIST et PRINTO.

4) Description détaillée:

- Bien sûr, on devra faire appel à la fonction EMPTY pour s'assurer que la liste n'est pas vide. Cet appel vérifiera que "1" désigne bien une tête de liste ou un objet assimilable à une tête de liste.

4.3.10. Fonction entière LAST (1).

1) Argument:

1: contient l'indice d'une tête de liste.

2) Description fonctionnelle:

- La fonction LAST renvoie à l'utilisateur l'indice du dernier membre de la liste dont la tête est désignée par "l".

- Si la liste est vide, elle renvoie un pointeur vers l'objet NONE. Cela indique évidemment qu'il n'existe pas de dernier membre dans la liste.

3) Position dans la chaîne des appels:

1. LAST appelle la fonction EMPTY.

2. Elle peut être appelée par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que le sous-programme PRINTO.

4) Description détaillée:

Comme dans le cas de la fonction FIRST, l'appel à EMPTY nous assurera que "l" désigne bien une tête de liste ou un objet assimilabla à une tête de liste.

4.3.11. Fonction entière SUC (i).

1) Argument:

i: contient l'indice d'un objet dont on désire connaître le successeur dans la liste.

2) Description fonctionnelle:

- Les attributs-système PRED et SUC d'un objet de simulation ne sont pas accessibles directement à l'utilisateur. En effet, nous avons voulu les protéger de manipulations inconsidérées. Cependant, le prédécesseur et le successeur d'un objet dans une liste sont des informations que le programmeur, à défaut de manipuler directement, doit pouvoir connaître à tout moment. C'est la raison d'être des fonctions PRED et SUC.

- En ce qui concerne la fonction SUC, elle renvoie à l'utilisateur l'indice de l'objet qui suit directement celui désigné par "i", dans la liste dont il est sensé faire partie.

- Si l'objet désigné par "i" est le dernier de la liste, la fonction SUC renvoie à l'utilisateur un pointeur vers l'objet NONE.

3) Position dans la chaîne des appels:

SUC peut être appelée par le programme principal ainsi que les sous-programmes et fonctions de l'utilisateur.

4.3.12. Fonction entière PRED (i).

1) Argument:

i: contient l'indice de l'objet dont on désire connaître le prédécesseur dans la liste.

2) Description fonctionnelle:

- La fonction PRED renvoie à l'utilisateur l'indice de l'objet qui précède directement celui désigné par "i", dans la liste dont il est sensé faire partie.
- Si l'objet désigné par "i" est le premier de la liste, la fonction PRED renvoie à l'utilisateur un pointeur vers l'objet NONE.

3) Position dans la chaîne des appels:

- PRED peut être appelée par le programme principal ainsi que les sous-programmes et fonctions de l'utilisateur.

4.3.13. Fonction entière CARDI (l).

1) Argument:

l: contient l'indice d'une tête de liste.

2) Description fonctionnelle:

La fonction CARDI renvoie à l'utilisateur le nombre de membres présents dans la liste dont la tête est désignée par "l". Bien sûr, si la liste est vide, elle renvoie la valeur nulle.

3) Position dans la chaîne des appels:

1. CARDI appelle, dans certains cas, le sous-programme ERROR.
2. Elle peut être appelée par le programme principal ainsi que les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- Il nous appartient de vérifier si "l" désigne bien une tête de liste ou un objet assimilable à une tête de liste. Si ce n'est pas le cas, un message d'erreur (numéro 310) est affiché et la simulation est arrêtée.

- Pour compter le nombre de membres de la liste, il nous faut parcourir séquentiellement toute la liste, en incrémentant une variable d'une unité pour chaque membre rencontré.

5) Remarque:

Si la fonction CARDI s'avérait d'usage trop fréquent, il serait avantageux d'ajouter aux têtes de listes un attribut supplémentaire contenant le nombre de membres présents, à chaque instant, dans la liste. Ce dernier devrait être mis à jour lors de chaque opération sur une liste. Par contre, la fonction CARDI se réduirait à la lecture d'un attribut-système (comme les fonctions SUC et PRED), évitant ainsi le balayage séquentiel de toute une liste.

4.4. STORAGES.

4.4.1. Sous-programme système VERIST (storag).

1) Argument:

storag: contient l'indice d'une station multiple.

2) Description fonctionnelle:

Lors de l'exécution d'un sous-programme LEAST (concernant la station multiple repérée par "storag"), le processus courant vient de libérer une certaine quantité "d'unités de serveur". Le sous-programme VERIST se charge de sélectionner et de réactiver (en tenant compte de leur priorité et de leur demande respectives) le plus grand nombre possible de transactions bloquées devant le serveur de cette station multiple désignée par "storag".

3) Position dans la chaîne des appels:

1. VERIST appelle les sous-programmes ACTIV, ADDCAP, ADDFLT, OUT ainsi que la fonction FIRST.
2. Il est appelé par le sous-programme LEAST.

4) Description détaillée:

- Dans la file d'attente associée à la station pointée par "storag", les transactions sont classées (par les soins de PRINTO) par ordre de priorité décroissante (la "plus" prioritaire se trouve en tête de la file). D'autre part, après l'exécution de l'ordre LEAST, supposons que N unités du serveur soient libres. Voilà brièvement décrit le contexte dans lequel s'exécute VERIST.
- VERIST réalise un parcours séquentiel de l'ensemble de la file d'attente, depuis la transaction la plus prioritaire jusqu'à la moins prioritaire.
- Pour chaque transaction rencontrée lors de ce balayage, les opérations suivantes sont effectuées:

  * si la demande ASK (mémorisée dans l'attribut-système PT de la transaction) est supérieure à N, on passe à la transaction suivante. Dans le cas contraire, on poursuit la séquence d'opérations;
  * on met à jour un ensemble important de variables statistiques;
  * on retire la transaction de la file d'attente;
  * on active immédiatement le processus correspondant à cette

transaction;

* enfin, on incrémente le contenu de la station de la demande ASK; ce qui revient, dans notre présente description, à décrémenter N de cette même valeur.

4.4.2. Sous-programme système PRINTO (i, l).

1) Arguments:

1. i: contient l'indice de l'objet-transaction que l'on désire placer en file d'attente. Dans le cadre de l'appel de PRINTO par ENTST ou ENTFAC, il s'agit du processus courant.

2. l: contient l'indice d'une station simple ou multiple. Celui-ci identifie la file d'attente associée à la station.

2) Description fonctionnelle:

- PRINTO insère la transaction désignée par "i" dans la file d'attente dont la tête de liste est désignée par "l" (nous avons déjà souvent indiqué qu'un storage, tout comme une facilité ou une région, était assimilable à une tête de liste).

- L'insertion s'effectue en tenant compte de la priorité de la transaction, de façon à respecter les règles suivantes:

  * dans une file d'attente associée à une station simple ou multiple, une transaction de priorité PP précède toutes celles de priorité srictement inférieure à PP;

  * pour deux transactions de même priorité:

    - la discipline d'attente reste classiquement FIFO si la priorité est positive (ou nulle). La première transaction introduite dans la file d'attente précède l'autre;

    - la discipline est LIFO si la priorité est négative. Dans ce cas, c'est la dernière transaction introduite dans la file qui précède l'autre.

3) Position dans la chaîne des appels:

1. PRINTO appelle les sous-programmes INTO, SPRECE, SFOLOW, ADDCAP ainsi que les fonctions FIRST et LAST.

2. Il est appelé par les sous-programmes ENTST et ENTFAC.

4) Description détaillée:

- Si la file d'attente est vide, il suffit d'insérer la transaction à l'aide de la procédure INTO.

- Soit FIRST la première transaction de la file d'attente. Si la priorité de FIRST est strictement inférieure à celle de la transaction désignée par "i", ou si les priorités sont négatives et identiques, la transaction "i" est placée en tête de la file d'attente.

- Si aucune de ces conditions n'est vérifiée, on parcourt la file d'attente séquentiellement, à partir de son dernier membre, jusqu'à ce que l'on trouve l'endroit où insérer la nouvelle transaction.

- Deux précisions s'imposent.

    * Si le premier test n'était pas programmé, l'insertion en tête de la file d'attente (que le test a pour but de détecter) ne pourrait pas se faire correctement.

    * Il est logique de parcourir la file à partir de son dernier élément. En effet, il est intéressant de minimiser le temps de balayage de la file. Or, le plus souvent, la discipline d'attente est FIFO. Dès lors, l'insertion s'effectue d'ordinaire à la queue de la file.

5) Remarque :

S'il n'y avait pas de manipulations statistiques (propres aux entités déduites de GPSSS), on pourrait envisager de rendre la procédure PRINTO accessible à l'utilisateur. Ce dernier pourrait alors gérer une file d'attente:

* soit selon la discipline FIFO, en utilisant INTO;

* soit selon une discipline avec priorités, en se servant de PRINTO.

4.4.3. Fonction entière NEWST (nomst, capac).

1) Arguments:

1. nomst: variable qui contient le nom de la station multiple. 15 caractères sont diponibles pour némoriser ce nom.

2. capac: variable qui contient la capacité de la station. Celle-ci correspond au nombre "d'unités de serveur" disponibles.

2) Description fonctionnnelle:

- Cette fonction permet à l'utilisateur de créer une nouvelle station multiple (ou "storage"). Elle réserve la place nécessaire à son implantation en mémoire et elle initialise l'ensemble de ses variables-système.

- Enfin, elle renvoie à l'utilisateur un indice qui lui permettra de référencer la station multiple qu'il vient de créer. C'est cet indice qu'il utilisera pour programmer l'entrée (ordre ENTST) et la sortie (ordre LEAST) d'une transaction dans la nouvelle station.

3) Position dans la chaîne des appels:

1. NEWST appelle, dans certains cas, le sous-programme ERROR.

2. Elle peut être appelée par le programme principal ainsi que par les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- La nouvelle station multiple créée prendra en mémoire la place d'une autre station détruite du même type. S'il n'en existe pas, l'implantation se fera à la suite de tous les autres objets (de toutes les classes) déjà créés auparavant, pour autant que la place-mémoire disponible y soit suffisante. Cette méthode d'allocation de place-mémoire est réalisée selon le même schéma que celui de la fonction NEW à laquelle nous renvoyons le Lecteur pour plus de précisions.

- L'attribut-système CAP prend comme valeur la capacité de la station multiple (valeur du paramètre capac).

- L'attribut-système CONTEN mémorise le "contenu" de la station. Remarquons que ce contenu ne représente pas le nombre de transactions en train d'être servies, mais bien le nombre "d'unités de serveur" présentement utilisées.

- La nouvelle station créée est chaînée dans la liste des stations multiples "actives" afin de permettre l'édition générique des statistiques automatiques de la simulation.

- Signalons encore que le nom de la station est mémorisé dans le vecteur-attribut (de trois positions) PTNAME. En FORTRAN Digital, cinq caractères ASCII peuvent être mémorisés dans un mot-mémoire. De là provient la limitation à 15 caractères du nom attribué à la nouvelle station créée. Cette remarque est valable également pour le nom des stations simples, des régions et des histogrammes.

- Le reste des attributs-système est destiné à mémoriser des résultats statistiques. Il serait fastidieux de définir la signification précise de chaque attribut-statistique des entités de SIMUFOR déduites de GPSSS. De même, il serait fastidieux d'indiquer systématiquement les modifications du contenu de ces attributs, ainsi que les endroits où celles-ci ont lieu. Nous pensons qu'une lecture attentive des sous-programmes, une fois informé des différents résultats statistiques auxquels nous nous sommes intéressés (cfr. fin du chap. 3), permet de se rendre compte de la manière dont ces statistiques sont récoltées. Le Lecteur désireux de suivre cette voie aurait avantage à lire, avant tout, l'introduction et la description des sous-programmes ADDINT, ADDFLT, ADDCAP qui constituent les outils de base de la récolte et du calcul des statistiques, de façon cumulative, dans SIMUFOR.

4.4.4. Sous-programme ENTST (sto, requi).

1) Arguments:

1. sto: contient l'indice permettant de référencer la station multiple dans laquelle la transaction désire entrer.

2. requi: indique le nombre d'unités du serveur demandées par la transaction pour pouvoir être prise en charge.

2) Description fonctionnelle:

- ENTST permet à une transaction (le processus courant) d'entrer dans une station multiple.

- Si le "contenu" de la station laisse assez d'unités de serveur libres pour satisfaire la demande (requi) de la transaction, cette dernière est servie immédiatement (après addition au contenu de la station, de la demande de la transaction).

- Dans le cas contraire, elle est placée en file d'attente (en tenant compte de sa priorité) devant le serveur. Quand son tour sera venu et que le nombre d'unités libres du serveur sera suffisant, la transaction bloquée pourra continuer son exécution.

3) Position dans la chaîne des appels:

1. ENTST appelle les sous-programmes EERROR, ADDINT, ADDCAP, PRINTO, RETURN et RESUME ainsi que le sous-programme assembleur SISAVE.

2. Il peut être appelé par le programme principal ainsi que par les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- Si "sto" ne désigne pas une station multiple, un message d'erreur (numéro 403) est affiché et la simulation s'arrête.

- Si la demande (en unités de serveur) de la transaction est plus importante que la capacité de la station, elle ne pourra jamais être servie. Dès lors, un message d'erreur (numéro 403) est affiché et la simulation est interrompue.

- Si le nombre d'unités libres du serveur est assez important pour satisfaire la demande de la transaction, celle-ci poursuit directement son exécution.

- Si cette dernière condition n'est pas vérifiée, la transaction est placée en file d'attente devant le serveur. Sa notice d'événement est retirée de la SQS et détruite. L'attribut-système PT devient inutile pour un processus qui se trouve en file d'attente. On l'utilise dès lors pour mémoriser la demande (en unités de serveur). Cette information doit en effet être conservée. Elle sera utilisée par la

procédure VERIST (cfr. plus haut).

- Notons que le placement en file d'attente s'effectue à l'aide de la procédure PRINTO qui tient compte de la priorité de la transaction.

- Enfin, le contrôle est rendu au processus qui se trouve à présent en tête de l'échéancier. De ce fait, ENTFAC peut être assimilé à un sous-programme de gestion des processus.

- La récolte des statistiques et la mise à jour des variables qui y sont associées représente une part importante du temps d'exécution du sous-programme ENTST. Il en sera de même pour LEAST, PRINTO, VERIST, ENTFAC, LEAFAC, ENTREG et LEAREG.

4.4.5. Sous-programme LEAST (sto, rlease).

1) Arguments:

1. sto: contient l'indice permettant de référencer la station multiple de laquelle la transaction désire sortir.

2. rlease: indique le nombre d'unités de serveur libérées par la sortie de la transaction.

2) Description fonctionnelle:

LEAST permet à la transaction courante de quitter la station multiple désignée par "sto" et dans laquelle elle est sensée se trouver. Ce faisant, elle libère "rlease" unités de serveur, permettant ainsi à certaines transactions, qui se trouvaient en file d'attente, d'être servies à leur tour.

3) Position dans la chaîne des appels:

1. LEAST appelle les sous-programmes EERROR, ADDCAP, ADDFLT, VERIST.

2. Il peut être appelé par le programme principal ainsi que par les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- Si "sto" ne désigne pas une station multiple, un message d'erreur (numéro 404) est affiché et la simulation s'arrête.

- De même, si la libération d'unités de serveur ("rlease") est plus importante que la capacité de la station, un message d'erreur (numéro 405) est affiché et la simulation est également interrompue.

- Le contenu de la station est décrémenté de la valeur "rlease". A ce moment, si ce contenu est devenu négatif, un message d'erreur (numéro 406) est encore affiché et la simulation est stoppée. Ces problèmes sont une conséquence directe du fait que l'on permet à une transaction

de libérer (à la sortie d'une station multiple) un nombre d'unités de serveur différent de celui qu'elle avait demandé (et obtenu) à l'entrée de la station.

- Enfin, après les mises à jour nécessaires des variables statistiques, LEAST fait appel à la procédure VERIST. Cette dernière s'occupe de servir le plus de transactions possible (à l'aide des unités de serveur libérées par le processus courant), en tenant compte de la priorité et de la demande de chacune des transactions candidates à la réactivation.

4.5. FACILITES.

4.5.1. Fonction entière NEWFAC (nomfac).

1) Argument:

nomfac : variable qui contient le nom de la facilité (maximum 15 caractères).

2) Description fonctionnelle:

- Cette fonction permet de créer, au cours de l'exécution, une nouvelle station simple (facilité). Elle réserve donc en mémoire la place nécessaire à son implantation et initialise ses différentes variables-système.
- Enfin, elle renvoie à l'utilisateur un indice qui lui permettra d'effectuer ultérieurement des opérations d'entrée et de sortie sur cette nouvelle facilité. Ces opérations sont respectivement ENTFAC et LEAFAC.

3) Position dans la chaîne des appels:

1. NEWFAC appelle, dans certains cas, le sous-programme EERROR.
2. Elle peut être appelée par le programme principal et les sous-programmes de l'utilisateur.

4) Description détaillée:

- La ressemblance entre la fonction NEW décrite auparavant et cette fonction NEWFAC nous permet de ne décrire que les lignes essentielles de cette dernière.
- NEWFAC vérifie d'abord s'il reste, en mémoire, assez de place pour implanter une nouvelle facilité. Si la place est insuffisante, un appel à EERROR signale à l'utilisateur ce manque de place et arrête la simulation. Par contre, si l'espace disponible est suffisant, elle réserve la place nécessaire, calcule l'indice destiné à l'utilisateur et initialise les différentes variables-système et statistiques de cette nouvelle station simple.

4.5.2. Sous-programme ENTFAC (fac).

1) Argument:

fac : contient l'indice qui permet de référencer la facilité. Cet indice a été renvoyé à l'utilisateur lorqu'il a créé cette facilité à l'aide de la fonction NEWFAC.

2) Description fonctionnelle:

ENTFAC permet à une transaction de pénétrer dans la facilité si celle-ci est libre. Si, par contre, elle est occupée, le processus courant qui décrit la transaction est passivé et mis dans une file d'attente jusqu'à la libération de cette facilité.

3) Position dans la chaîne des appels:

1. ENTFAC peut appeler les sous-programmes PRINTO, RETURN, RESUME et EERROR ainsi que le sous-programme assembleur SISAVE.
2. Il peut être appelé par le programme principal et les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- Ce sous-programme vérifie en premier lieu si l'indice fourni par le paramètre "fac" fait référence à un objet de la classe "facilité".
- Si l'indice ne référence pas une station simple, un message d'erreur est envoyé à l'utilisateur et la simulation est arrêtée.
- Par contre, si l'indice est correct, ENTFAC vérifie si la transaction qui a généré cet appel se trouve déjà dans cette facilité, auquel cas un message d'erreur est envoyé à l'utilisateur et la simulation est stoppée.
- Enfin, si la facilité est libre, la transaction est autorisée à progresser après la mise à jour de variables statistiques, tandis que si la facilité est occupée, le processus représentant cette transaction (c'est-à-dire le processus courant) est chaîné dans une liste associée à la station simple, son point de réactivation est sauvé, sa notice d'événement est retirée de la SQS et un appel au sous-programme RESUME est généré.

4.5.3. Sous-programme LEAFAC (fac).

1) Argument:

fac: indice qui permet de référencer la facilité (voir argument du sous-programme ENTFAC).

2) Description fonctionnelle:

LEAFAC permet à la transaction décrite par le processus courant de sortir de la facilité dans laquelle elle se trouvait.

3) Position dans la chaîne des appels:

1. LEAFAC appelle les sous-programmes ADDCAP, ADDFLT, OUT, ACTIV, EERROR ainsi que la fonction FIRST.

2. Il peut être appelé par le programme principal et les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- Après avoir vérifié si l'indice passé comme paramètre est correct (c'est-à-dire s'il référence une facilité), LEAFAC compare l'indice du processus courant et l'indice du processus qui se trouve à l'intérieur de la station.

- Si ces deux indices sont différents, l'appel à LEAFAC est erroné et un message d'erreur est envoyé à l'utilisateur.

- Par contre, s'ils sont égaux, LEAFAC met à jour les variables statistiques concernant les facilités.

- Si la file d'attente associée à cette station simple n'est pas vide, LEAFAC sélectionne le premier processus de cette file et met à jour les statistiques grâce au sous-programme ADDCAP. Ce premier processus est alors retiré de cette file à l'aide du sous-programme OUT et est chaîné dans la SQS par un appel à ACTIV.

4.6. GROUPES.

4.6.1. Fonction entière NEWGR (capac).

1) Argument:

capac: précise la capacité du groupe.

2) Description fonctionnelle:

NEWGR permet à l'utilisateur de créer un nouveau groupe. Elle réserve donc en mémoire la place nécessaire à l'implantation de ce nouveau groupe et en initialise les différentes variables-système. Enfin, elle renvoie à l'utilisateur un indice qui doit être utilisé lorsqu'une transaction veut entrer dans ce nouveau groupe.

3) Position dans la chaîne des appels:

1. NEWGR appelle dans certains cas le sous-programme EERROR.
2. Elle peut être appelée par le programme principal, les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

Cette fonction est tout à fait similaire aux fonctions NEW, NEWST, NEWFAC,... déjà décrites: elle permet la recherche de la place nécessaire à l'implantation des variables-système, la réservation de cet espace mémoire, le calcul de l'indice destiné à l'utilisateur et l'initialisation des différentes variables.

4.6.2. Sous-programme JOIN (gr).

1) Argument:

gr: indice qui permet de référencer un groupe.

2) Description fonctionnelle:

JOIN permet à la transaction décrite par le processus courant d'entrer dans le groupe désigné par "gr". Si le nombre de transactions dans le groupe n'est pas supérieur ou égal à la capacité du groupe, cette transaction est mise en attente; le processus courant est donc passivé et chaîné dans la liste d'attente associée à ce groupe.

3) Position dans la chaîne des appels:

1. JOIN appelle, dans certains cas, les sous-programmes EERROR, RETURN, INTO, RESUME, ACTIV, OUT, HOLD, le sous-programme assembleur SISAVE ainsi que la fonction FIRST.

2. Il peut être appelé par le programme principal et les sous-programmes décrivant les processus de l'utilisateur.

4) Description détaillée:

- Comme la plupart des sous-programmes décrivant les opérations effectuées sur des objets basés sur les concepts de GPSS, JOIN vérifie en premier lieu la validité de l'indice passé comme paramètre.

- Si celui-ci n'est pas correct, c'est-à-dire s'il ne référence pas un groupe, un message d'erreur est envoyé à l'utilisateur et la simulation est arrêtée.

- Par contre, si l'indice est correct, JOIN met à jour la variable qui comptabilise le nombre de transactions qui se trouvent dans ce groupe.

- Si le nombre de transaction est inférieur à la capacité du groupe, le processus courant (qui représente la transaction) est passivé: son adresse de réactivation est sauvée, sa notice d'événement est retirée de la SOS et ce processus est chaîné dans la liste associée au groupe. Enfin, un appel à RESUME sélectionne et active le nouveau processus courant.

- Par contre, si le nombre de transactions est supérieur ou égal à la capacité de ce groupe, il est remis à zéro, les processus qui se trouvent dans la file d'attente en sont extraits et sont cédulés à l'heure courante; ceci est réalisé grâce à des appels aux sous-programmes ACTIV et OUT.

4.7. REGIONS.

4.7.1. Fonction entière NEWREG (nomreg).

1) Argument:

nomreg: variable qui contient le nom de la région (maximum de 15 caractères).

1) Description fonctionnelle:

- NEWREG permet à l'utilisateur de créer une nouvelle région.
- Cette fonction réserve donc la place-mémoire nécessaire à l'implantation de cette nouvelle entité. Elle initialise ses différents attributs-système.
- Elle renvoie enfin à l'utilisateur l'indice de cette région, indice qui permettra à une transaction d'entrer dans la région (par l'ordre ENTREG) et d'en sortir (par l'ordre LEAREG).

3) Position dans la chaîne des appels:

- NEWREG appelle, dans certains cas, le sous-programme ERROR.
- Elle peut être appelée par le programme principal ainsi que par les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- La technique d'allocation de place-mémoire à un nouvel objet créé a déjà été détaillée, par exemple dans la description de la fonction NEW. Nous n'y reviendrons donc pas.
- Comme en ce qui concerne les autres types d'entités déduites de GPSSS, la nouvelle région créée est insérée dans une liste (par son attribut-système LIST) des régions "actives".
- Le reste du code de NEWREG consiste dans l'initialisation des attributs-système qui servent à la récolte et au calcul des statistiques.

4.7.2. Sous-programme ENTREG (reg).

1) Argument:

reg: contient l'indice de la région dans laquelle on désire entrer.

2) Description fonctionnelle:

ENTREG permet à la transaction courante d'entrer dans la région désignée par "reg".

3) Position dans la chaîne des appels:

1. ENTREG appelle les sous-programmes EERROR et ADDCAP.
2. Elle peut être appelée par le programme principal ainsi que par les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- Si l'objet désigné par "reg" n'est pas une région, une erreur (numéro 702) est générée et la simulation est interrompue.
- Nous avons déjà indiqué au chap. 3 l'utilité purement statistique de la notion de région. De fait, le code de ENTREG consiste presque exclusivement en la manipulation de variables statistiques.

4.7.3. Sous-programme LEAREG (reg).

1) Argument:

reg: contient l'indice de la région dont on désire sortir.

2) Description fonctionnelle:

LEAREG permet à la transaction courante de sortir de la région désignée par "reg" et dans laquelle elle est sensée se trouver.

3) Position dans la chaîne des appels:

1. LEAREG appelle les sous-programmes EERROR, ADDFLT et ADDCAP.
2. Elle peut être appelée par le programme principal ainsi que par les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- Si l'objet désigné par "reg" n'est pas un région, une erreur (numéro 704) est générée et la simulation est arrêtée.
- LEAREG poursuit son exécution par la mise à jour des statistiques concernant la région que le processus courant vient de quitter.

4.8. HISTOGRAMMES.

4.8.1. Fonction entière NEWHIS (nomhis, nbint, borinf, taille).

1) Arguments:

- nomhis: permet de donner un nom à l'histogramme (maximum 15 caractères);
- nbint: précise le nombre d'intervalles désirés;
- borinf: donne la borne inférieure de l'histogramme;
- taille: indique la taille unique pour tous les intervalles de cet histogramme.

2) Description fonctionnelle:

NEWHIS permet à l'utilisateur de créer un nouvel histogramme. Elle réserve donc en mémoire la place nécessaire à son implantation et initialise ses différentes variables-système. Enfin, elle renvoie à l'utilisateur un indice qui devra être utilisé lorsqu'il voudra saisir (à l'aide de ADDHIS) les données de l'histogramme désiré.

3) Position dans la chaîne des appels:

1. NEWHIS appelle dans certains cas le sous-programe EERROR.
2. Elle peut être appelée par le programme principal et les sous-programmes de l'utilisateur.

4) Description détaillée:

Cette fonction est tout à fait similaire aux fonctions NEW, NEWST, NEWFAC, ... déjà décrites: elle permet la recherche de la place nécessaire, la reservation de cet espace mémoire, le calcul de l'indice destiné à l'utilisateur et l'initialisation des différentes variables.

4.8.2. Sous-programme ADDHIS (hist, val).

1) Arguments:

- hist: contient l'indice qui référence un histogramme;
- val: contient la valeur à classer.

2) Description fonctionnelle:

ADDHIS réalise la saisie et le traitement des valeurs dont l'histogramme doit être dressé.

3) Position dans la chaîne des appels:

1. ADDHIS appelle dans certains cas le sous-programme EERROR.
2. Il peut être appelé par le programme principal, les sous-programmes et fonctions de l'utilisateur.

4) Description détaillée:

- ADDHIS vérifie en premier lieu si l'indice fourni par la variable "hist" fait référence à un objet de la classe "histogramme".
- Si l'indice ne référence pas un histogramme, un message d'erreur est envoyé à l'utilisateur et la simulation est arrêtée.
- Par contre, si l'indice est correct, ADDHIS met à jour ses différents compteurs et calcule dans quel intervalle doit être comptabilisée la valeur fournie par "val".

4.8.3. Sous-programme PRTHIS (hist).

1) Argument:

hist: contient l'indice qui permet de référencer un histogramme.

2) Description fonctionnelle:

PRTHIS réalise l'impression de l'histogramme dont l'indice est fourni par "hist".

3) Position dans la chaîne des appels:

PRTHIS peut être appelé par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que par le sous-programme HISREP.

4) Description détaillée:

PRTHIS ne contient que des impressions.

4.8.4. Sous-programme HISREP.

1) Arguments: nihil.

2) Description fonctionnelle:

HISREP réalise l'impression de tous les histogrammes générés lors de la simulation.

3) Position dans la chaîne des appels:

1. HISREP appelle le sous-programme PRTHIS.
2. Il peut être appelé par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que le sous-programme GENREP.

4) Description détaillée:

HISREP parcourt la chaîne qui relie tous les histogrammes et commande leur impression par des appels successifs au sous-programme PRTHIS.

4.9. NOMBRES ALEATOIRES.

Nous fournissons à l'utilisateur éventuel de SIMUFOR, en plus des sous-programmes et fonctions nécessaires au contrôle d'une simulation, une série de fonctions qui génèrent, lors d'appels successifs, un ensemble de nombres pseudo-aléatoires appartenant à des distributions particulières.

Ces fonctions de génération de nombres aléatoires sont toutes construites autour d'une fonction de base RAND et peuvent être appelées par tous les sous-programmes et toutes les fonctions de l'utilisateur.

Non seulement ces fonctions renvoient un nombre pseudo-aléatoire, mais en outre, elles mettent à jour le germe "u" (variable entière) qui doit être passé comme paramètre lors de chaque appel.

Nous nous permettons, pour ces fonctions, de ne donner qu'une brève "description fonctionnelle". Une description plus détaillée, une justification et une validation des approximations employées sortiraient en effet du cadre du présent travail. Nous renvoyons donc le Lecteur à l'étude de DOWNHAM [4] pour la fonction RAND. Le Lecteur trouvera en outre de longs développements en ce qui concerne les autres fonctions de génération de nombres aléatoires dans l'ouvrage de NAYLOR et al. [14].

4.9.1. Fonction réelle RAND (u).

Elle génère des nombres pseudo-aléatoires uniformément distribués sur [0,1[.

4.9.2. Fonction booléenne DRAW (a,u).

Elle fournit une valeur booléenne qui est "VRAIE" (TRUE) avec une probabilité "a" et "FAUSSE" (FALSE) avec une probabilité "1-a" ($a \in R$). Cette valeur est toujours "VRAIE" lorsque "a" est supérieur ou égal à 1, tandis qu'elle est toujours "FAUSSE" lorsque "a" est inférieur ou égal à 0.

4.9.3. Fonction réelle UNIF (a,b,u).

Elle génère des nombres pseudo-aléatoires uniformément distribués sur [a,b[(a et b ∈ R, a ⩾ b). Une erreur qui entraîne l'arrêt de la simulation est détectée lorsque "b ⩽ a".

4.9.4. Fonction entière RANDIN (a,b,u).

Elle renvoie un nombre entier parmi "a, a+1, a+2, ..., b-2, b-1, b"; chaque valeur possède une probabilité égale d'être tirée.

4.9.5. Fonction réelle NEGEXP (a,u).

La fonction génère une valeur d'une distribution exponentielle négative de moyenne "1/a" définie par "-ln(x)/a" où "x" est un nombre aléatoire de base obtenu par un appel à RAND.

4.9.6. Fonction réelle NORMAL (a,b,u).

Cette fonction génère des variables pseudo-aléatoires d'une distribution normale de moyenne "a" et d'écart-type "b".

Sa fonction de répartition est :

$$f(x) = \frac{1}{b\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-a}{b}\right)^2} \qquad -\infty < x < \infty$$

4.9.7. Fonction réelle POISSN (a,u).

La valeur renvoyée par cette fonction est un nombre aléatoire d'une distribution de Poisson de paramètre "a".

$$f(x) = e^{-at} \frac{(at)^x}{x!}$$

4.9.8. Fonction réelle GAMMA (k,a,u).

Elle génère des variables aléatoires d'une distribution Gamma dont la fonction de répartition est

$$f(x) = \frac{a^k \, x^{(k-1)} \, e^{-ax}}{(k-1)!} \qquad \text{où } a>0,\ k>0,\ x \geq 0$$

4.9.9. Fonction entière HYPGEO (npop,nech,p,u).

Elle génère un nombre aléatoire de distribution hypergéométrique H(Np, Nq, x). et dont la fonction de densité est:

$$f(x) = \frac{\binom{N_p}{x}\binom{N_q}{n-x}}{\binom{N}{n}}$$

$0 \leq x \leq N_p$
$0 \leq n-x \leq N_q$
$n_p + n_q = N$

$N = n_{pop}$
$n = n_{ech}$

4.9.10. Fonction entière BINOM (n,p,u).

Elle génère une variable aléatoire définie par le nombre de succès dans un ensemble de "n" tirages indépendants de Bernouilli. "p" est la probabilité d'un succès pour chaque tirage.

$$f(x) = \binom{n}{x} p^x q^{n-x}$$

$q = 1-p$
$x =$ entier $E[0, n]$

4.10. DEBUGGING, TRACAGE ET ERREURS.

4.10.1. Sous-programme système WERROR (n).

1) Argument:

n: indique le numéro de l'avertissement.

2) Description fonctionnelle:

- WERROR regroupe tous les messages d'avertissement qui peuvent être envoyés à l'utilisateur.
- Un appel à WERROR provoque l'impression du message d'avertissement correspondant au numéro passé comme paramètre.
- Remarquons que 5 messages d'avertissement au maximum peuvent être générés: un dépassement de ce nombre provoque immédiatement l'arrêt de la simulation.

3) Position dans la chaîne des appels:

WERROR peut être appelé par les sous-programmes ACTIV, KILL, PACTIV, HOLD, CANCEL.

4.10.2. Sous-programme système EERROR (n).

1) Argument:

n: indique le numéro du message d'erreur.

2) Description fonctionnelle:

- EERROR regroupe tous les messages d'erreur qui peuvent être envoyés à l'utilisateur.
- Un appel à ce sous-programme provoque l'impression du message d'erreur correspondant au numéro fourni en paramètre et arrête le déroulement de la simulation.

3) Position dans la chaîne des appels:

EERROR peut être appelé par les fonctions CLASS, NEW, IDLE, CARDI, EVTIME, NEWST, NEWFAC, NEWGR, NEWREG, NEWHIS, RANDIN ainsi que les sous-programmes RETURN, ACTIV, PACTIV, REACT, PREACT, KILL, OUT, PRECED, INTO, EMPTY, ENTST, LEAST, ENTFAC, LEAFAC, ENTREG, LEAREG, JOIN, ADDHIS, PRTHIS, INISTO, INIFAC, INIREG.

4.10.3. Sous-programme TRACE.

1) Arguments: nihil.

2) Description fonctionnelle:

- TRACE permet de suivre pas à pas le déroulement de la simulation: création des différents objets et processus ou transactions, activation et destruction des processus, ... ainsi que toutes les opérations concernant les storages, les facilités et les groupes (création, entrée et sortie).

- L'utilisateur a le choix entre 3 niveaux de trace:

  1. pas de trace (call untrac) (option par défaut),
  2. une trace intermédiaire (call trace),
  3. une trace entrecoupée de "dumps" de la mémoire pseudo-dynamique et de parcours de la SQS chaque fois que le sous-programme RESUME est appelé c'est-à-dire approximativement chaque fois qu'un nouveau processus est défini comme processus courant (call tracex).

- Ce choix peut être modifié tout au long de la simulation.

3) Position dans la chaîne des appels:

TRACE et ses entrées (UNTRAC et TRACEX) peuvent être appelés par le programme principal, les sous-programmes et les fonctions de l'utilisateur.

4.10.4. Sous-programme SUIVRE.

1) Arguments: nihil.

2) Description fonctionnelle:

SUIVRE réalise un parcours de la SQS; pour chaque processus (actif ou suspendu), il imprime:

* son numéro d'ordre dans la SQS,
* son indice,
* l'heure de sa prochaine phase active cédulée,
* son numéro de classe,
* l'indice de son prédécesseur dans la SQS,

* l'indice de son successeur dans la SOS,

* la valeur de son point de réactivation.

3) Position dans la chaîne des appels:

SUIVRE peut être appelé par le programme principal et les sous-programmes de l'utilisateur ainsi que le sous-programme RESUME.

4) Description détaillée:

- SUIVRE parcourt la chaîne formée par l'ensemble des notices d'événement des divers processus cédulés.
- Il accède ainsi directement aux informations telles que le numéro d'ordre, l'indice et l'heure d'événement. Les autres informations (numéro de classe, indices des prédécesseur et successeur, pointeur de réactivation) sont obtenues par l'intermédiaire du pointeur PT qui relie la notice d'événement à l'objet-processus.

4.10.5. Sous-programme DUMPS.

1) Arguments: nihil.

2) Description fonctionnelle:

Ce sous-programme réalise l'impression:

1. de quelques informations générales
   (indice de l'objet NONE, indice de la SOS, indice du processus courant, heure courante de la simulation);
2. des caractéristiques de chaque classe déclarée
   (numéro de la classe, sa taille, l'adresse du début du code qui lui est associé ainsi que l'indice qui permet d'atteindre sa liste des objets détruits);
3. du contenu de la zone d'implémentation des divers objets de la simulation
   (numéro de la case mémoire, indice pour la référencer, contenu sous forme entière et contenu sous forme octale).

3) Position dans la chaîne des appels:

DUMPS peut être appelé par le programme principal, les sous-programmes et fonctions de l'utilisateur ainsi que le sous-programme RESUME.

4) Description détaillée:

- Les informations générales (1) et celles relatives au descripteur de classe (2) sont obtenues de façon tout à fait classique: les premières par consultation de la variable portant leur nom, les secondes par un simple indiçage FORTRAN.

- L'impression du contenu de la zone d'implantation des objets de la simulation est obtenue d'une manière tout aussi simple: au lieu de considérer (et cela compliquerait cette impression) que la zone d'adressage pseudo-dynamique (commun ZAD) est formée par un ensemble d'occurence des variables-système et variables-utilisateur de différentes classes, on calcule les deux indices qui permettent, à partir d'un tableau fictif (en l'occurence le tableau CLASS), d'atteindre le début et la fin de la partie occupée dans le commun ZAD; de nouveau, un simple indiçage itératif permet alors d'imprimer le contenu de tout ce bloc.

4.11. STATISTIQUES.

Nous croyons opportun de présenter quelques remarques à propos des statistiques que nous calculons automatiquement dans SIMUFOR. En concevant le système SIMUFOR, nous désirions implémenter en FORTRAN la grande majorité des concepts et des primitives du langage SIMULA. Nous croyons avoir atteint cet objectif.

Nous avions également conscience, d'une part de la difficulté de programmer des problèmes de réseaux de files d'attente en SIMULA, et d'autre part de la fréquence de ces problèmes. Nous avons donc estimé souhaitable d'incorporer à notre système les notions et les entités de GPSSS.

L'étape suivante consistait alors à calculer automatiquement des statistiques à propos de ces entités (qui se prêtent particulièrement bien à ce genre d'exercice). Malheureusement, le temps nous était compté et nous nous sommes donc contentés de récolter des statistiques à propos de toutes les données qui respectent les trois conditions suivantes:

- possibilité de calcul cumulatif qui évite la mémorisation de masses trop importantes d'informations;
- accès relativement aisé à l'information;
- absence de concepts statistiques compliqués qui nécessitent une étude théorique préalable ou des traitements particulièrement lourds.

Nous n'avons pas la prétention d'affirmer que les statistiques que nous calculons sont les plus importantes ou les plus pertinentes: elles ne sont qu'une ébauche de ce qui pourrait être fait. L'évolution de SIMUFOR passe naturellement par des tests de stationnarité et le calcul d'intervalles de confiance, pour ne citer que quelques développements statistiques possibles. Cependant, nous croyons que les résultats que nous fournissons permettent une vue assez claire, à défaut d'être précise et rigoureuse, du déroulement d'une simulation de réseaux de files d'attente.

Le principal problème, qui n'est pas résolu, reste la détermination de la stationnarité (ou la non-stationnarité) d'un phénomène stochastique. Les écarts-types que nous calculons ne donnent qu'une idée assez vague de cette

information. L'utilisateur averti de SIMUFOR devra donc, au stade actuel du développement de notre système, faire preuve d'un certain doigté ou programmer lui-même les tests de stationnarité. De plus, il peut évidemment programmer le calcul de toutes les statistiques particulières qu'il estime nécessaires.

Après avoir fait cette mise au point, nous nous intéresserons d'une façon pragmatique, et non plus critique, aux statistiques que nous calculons. Il faut distinguer les informations ponctuelles et les informations rapportées à l'échelle du temps.

1) Informations ponctuelles.

- Le nombre d'unités d'un serveur (d'une station multiple particulière) demandées par une transaction est une information ponctuelle par excellence. Elle est récoltée à chaque entrée d'une transaction dans le "storage" correspondant. Il s'agit ici d'une information entière. C'est la seule; toutes les autres sont réelles (en effet, pour SIMUFOR, le temps simulé est une valeur réelle).

- Le temps de service des transactions par un serveur est une autre information ponctuelle. On mémorise l'heure du début du service dans l'attribut-système PTSTAT de la transaction. Cette donnée est retranchée de l'heure de la fin de service; on obtient ainsi une information à propos du temps de service.

- Le temps d'attente des transactions dans une file est encore une information ponctuelle. Elle s'obtient par différence entre l'heure d'insertion de la transaction dans la file et l'heure de son retrait.

Dans ces trois cas, on obtient un échantillon de valeurs ponctuelles (observations). On peut en calculer:

- la valeur minimale (min);
- la valeur maximale (max);
- la moyenne des valeurs (av);
- la moyenne des carrés des valeurs (std);
- la taille de l'échantillon.

Ces cinq informations peuvent s'obtenir de manière cumulative. Il suffit d'initialiser les cinq variables de la façon suivante:

- min = $+\infty$,
- max = 0 (ou 0.0) ,

- av = 0.0 ,
- std = 0.0 ,
- nb = 0 .

Lorsqu'apparaît une nouvelle observation (obs), on la traite directement par l'appel:

CALL ADDINT (av, std, nb, min, max, obs),

si la valeur observée est entière, ou par:

CALL ADDFLT (av, std, nb, min, max, obs),

si la valeur observée est réelle. A tout instant de la simulation, si on considère l'échantillon des valeurs qui ont déjà été observées:

- min représente la valeur minimale;
- max représente la valeur maximale;
- av représente la valeur moyenne;
- std représente la moyenne des carrés des valeurs;
- nb représente la taille de l'échantillon.

L'écart-type (ec-typ) de l'échantillon s'obtient aisément à partir de la formule:

$$\text{ec-typ} = \text{var}^{\frac{1}{2}}$$

où $\text{var} = \text{std} - \text{av}^2$.

2) Informations rapportées à l'échelle du temps.

- L'évolution du nombre des transactions bloquées dans une file d'attente est typiquement une information rapportée à l'échelle du temps. C'est un processus stochastique.
- Il en est de même de l'évolution du nombre d'unités occupées du serveur d'une station multiple.

Dans le cas d'une simulation à événements discrets, l'évolution d'une telle information se représente, sur l'échelle du temps, par une figure du type suivant: voir fig. 4-4.

La valeur maximale et la valeur minimale de l'information sont aussi aisées

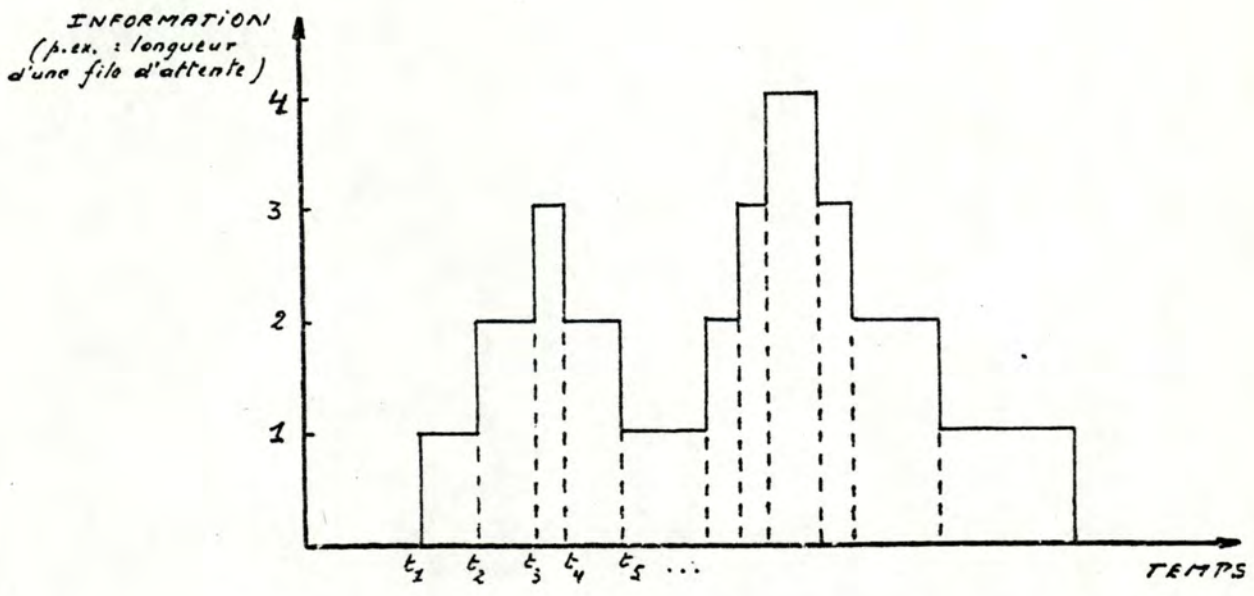


Figure 4-4: Evolution d'une information temporelle.

à obtenir que dans le cas d'un échantillon de valeurs ponctuelles.

Par contre, la notion de moyenne et celle d'écart-type sont plus difficiles à cerner que dans ce cas.

Nous utiliserons la notion de moyenne ergodique, plus naturelle dans le cas d'un processus stochastique. Celle-ci est définie comme l'intégrale de l'information depuis le début jusqu'à la fin de l'observation, divisée par la durée de celle-ci. Dans le cas d'une simulation à événements discrets, le nombre des changements de la valeur de l'information est fini. De cette façon, le calcul de l'intégrale se résume à l'addition des surfaces de petits rectangles, comme on peut le voir sur la figure 4-4. Notons que la moyenne ergodique n'a de sens que dans le cas d'un processus stochastique stationnaire. L'information que nous fournissons devra donc être considérée avec prudence.

Pour donner une idée de la dispersion de l'information autour de la moyenne (ergodique) calculée, nous avons inventé un indice de dispersion que nous avons appelé "écart-type ergodique". Il a été déduit de l'écart-type classique par le même cheminement que celui qui permet de déduire la moyenne ergodique de la

moyenne classique. La moyenne ergodique et notre "écart-type ergodique" peuvent aussi être calculés de manière cumulative. Il suffit à nouveau d'initialiser cinq variables de la façon suivante:

- cpmin = $+\infty$,
- cpmax = 0 ,
- cpav = 0.0 ,
- cpstd = 0.0 ,
- tptot = 0.0 .

Lorsque l'information que l'on observe change de valeur, on met dans une variable (tpobs) la longueur du temps durant laquelle l'information n'a pas changé (c'est-à-dire la différence entre l'heure courante et l'heure du précédent changement de valeur). On mémorise également (obs) la valeur de l'information avant le changement que l'on vient d'enregistrer. Ces deux données représentent une observation que l'on traite directement par l'appel:

```
CALL ADDCAP (cpav,cpstd,tptot,cpmin,cpmax,obs,tpobs).
```

De cette manière, à l'issue de chaque changement de valeur de l'information, et sur toute la période d'observation déjà couverte (représentée par tptot):

- cpmin représente le valeur minimale de l'information;
- cpmax en représente la valeur maximale;
- cpav en représente la moyenne ergodique;
- cpstd représente la moyenne ergodique du carré de l'information (si l'on ose dire).

L'écart-type "ergodique" s'obtient de la même manière que l'écart-type classique, à partir de cpstd et cpav, par la formule:

$$\text{ec-typ} = \text{cpvar}^{\frac{1}{2}} ,$$

où $\quad \text{cpvar} = \text{cpstd} - \text{cpav}^{2}$.

4.11.1. Sous-programme ADDFLT (moy, var, compt, min, max, neuf).

1) Arguments:

1. moy: variable (réelle) contenant la moyenne des valeurs déjà observées.
2. var: variable (réelle) contenant la moyenne des carrés des valeurs déjà observées.
3. compt: variable (entière) contenant le nombre de valeurs déjà observées.
4. min: variable (réelle) contenant la plus petite valeur observée.
5. max: variable (réelle) contenant la plus grande valeur observée.
6. neuf: variable (réelle) contenant la valeur de la nouvelle observation.

2) Description foctionnelle:

- Le sous-programme ADDFLT permet d'ajouter une nouvelle valeur observée à un échantillon de valeurs réelles. Il assure l'ajustement du minimum et du maximum des valeurs déjà observées. Il calcule les nouvelles moyennes des valeurs et des carrés des valeurs. Enfin, il incrémente d'une unité la taille de l'échantillon.
- Il pourrait être utilisé par un programmeur désireux d'effectuer les mêmes opérations à peu de frais.

3) Position dans la chaîne des appels:

ADDFLT est appelé par les sous-programmes LEAFAC, VERIST, LEAST et LEAREG.

4) Description détaillée:

- Les nouveaux maximum et minimum sont déterminés par comparaison des anciens avec la nouvelle valeur observée.
- La nouvelle moyenne se calcule par la formule:

$$moy = \frac{(compt * moy) + neuf}{compt + 1}$$

- La nouvelle moyenne des carrés, quant à elle, se calcule par la formule:

$$var = \frac{(compt * var) + neuf^{2}}{compt + 1}$$

- L'incrémentation de compt ne peut s'effectuer qu'après ces deux calculs.

4.11.2. Sous-programme ADDCAP (cpmoy, cpvar, tptot, cpmin, cpmax, cpneuf, tpneuf).

1) Arguments:

1. cpmoy: variable (réelle) contenant la moyenne ergodique de l'information sur la période d'observation "tptot".
2. cpvar: variable (réelle) contenant la moyenne ergodique du carré de l'information, sur la même période.
3. tptot: variable (réelle) contenant la durée de l'observation de l'information, depuis le début jusqu'au changement de valeur qui a provoqué l'appel précédent à ADDCAP.
4. cpmin: variable (entière) contenant la valeur minimale de l'information sur la période d'observation "tptot".
5. cpmax: variable (entière) contenant la valeur maximale de l'information sur cette période.
6. cpneuf: variable (entière) contenant la valeur de l'information depuis l'appel précédent à ADDCAP jusqu'à l'appel actuel.
7. tpneuf: variable (réelle) contenant la durée de temps qui a séparé ces deux appels successifs.

2) Description fonctionnelle:

- Supposons que nous ayons affaire à une information entière qui varie par changements discrets de valeur (on dit également d'état) au cours du temps. Chacun de ces changements provoque l'appel à ADDCAP.
- En tenant compte de la valeur de l'information entre le changement d'état qui a provoqué l'appel actuel de ADDCAP et le précédent, ce sous-programme réalise l'ajustement:

  * des valeurs minimale et maximale de l'information;
  * des moyennes ergodiques de l'information et du carré de l'information;
  * de la durée de la période d'observation déjà prise en compte.

3) Position dans la chaîne des appels:

ADDCAP est appelé par les sous-programmes LEAFAC, VERIST, LEAST, LEAREG, PRINTO, ENTST, ENTREG, STOREP, FACREP et REGREP.

4) Description détaillée:

- Les nouveaux minimum et maximum sont toujours déterminés par comparaison entre leurs anciennes valeurs et la valeur de "cpneuf".
- La nouvelle moyenne ergodique se calcule par la formule:

$$cpmoy = \frac{(tptot * cpmoy)+(cpneuf * tpneuf)}{(tptot + tpneuf)}$$

- La nouvelle moyenne ergodique du carré de l'information se calcule, elle, par la formule:

$$cpvar = \frac{(tptot * cpvar)+(cpneuf^2 * tpneuf)}{(tptot + tpneuf)}$$

- Nous pouvons remarquer la similitude qui existe entre ces formules et celles utilisées dans le sous-programme ADDFLT. Ici, les observations de l'information sont simplement pondérées par la durée de la période durant laquelle l'information est restée constante.
- Enfin,l'incrémentation de "tptot" ne peut se faire qu'après l'accomplissement de tous les calculs précédents.

4.11.3. Sous-programme ADDINT (moy, var, compt, min, max, neuf).

1) Remarque:

ADDINT ne diffère de ADDFLT que par le fait qu'il travaille sur un échantillon de valeurs entières. Par conséquent, min, max et neuf représentent des variables entières. Pour le reste, nous renvoyons le Lecteur à la description de ADDFLT. Il est à noter que la manipulation des types de variables est très délicate en FORTRAN. C'est d'ailleurs la raison pour laquelle nous avons dû écrire deux sous-programmes différents qui réalisent fonctionnellement le même traitement.

2) Position dans la chaîne des appels:

ADDINT est appelé par ENTST.

4.11.4. Sous-programme STOREP.

1) Arguments: nihil.

2) Description fonctionnelle:

Le sous-programme STOREP provoque l'impression de toutes les statistiques (calculées automatiquement par SIMUFOR) disponibles à propos de l'ensemble de stations multiples de la simulation. Les données affichées sont divisées en six grandes classes d'information qui ont été définies à la fin du chap. 3.

3) Position dans la chaîne des appels:

1. STOREP appelle le sous-programme ADDCAP.
2. Il peut être appelé par le programme principal, les sous-programmes de l'utilisateur ainsi que le sous-programme GENREP.

4) Description détaillée:

- Pour chaque classe d'information, le sous-programme STOREP parcourt la chaîne qui relie l'ensemble des stations multiples "actives" au moment de l'appel, et imprime une ligne de résultats pour chacune d'entre elles.
- Nous nous sommes évidemment efforcés d'éditer les résultats de façon à en rendre la lecture la plus aisée possible.

4.11.5. Sous-programme FACREP.

1) Arguments: nihil.

2) Description fonctionnelle:

Le sous-programme STOREP provoque l'impression de toutes les statistiques (calculées automatiquement par SIMUFOR) disponibles à propos de l'ensemble de stations simples de la simulation. Les données affichées sont divisées en quatre grandes classes d'information qui ont été définies à la fin du chap. 3.

3) Position dans la chaîne des appels:

1. FACREP appelle le sous-programme ADDCAP.
2. Il peut être appelé par le programme principal, les sous-programmes de l'utilisateur ainsi que le sous-programme GENREP.

4) Description détaillée:

- Pour chaque classe d'information, le sous-programme FACREP parcourt la chaîne qui relie l'ensemble des stations simples "actives" au moment de l'appel, et imprime une ligne de résultats pour chacune d'entre elles.

- L'édition des résultats s'effectue selon le même schéma que celui qui a été mis au point pour les stations multiples.

4.11.6. Sous-programme REGREP.

1) Arguments: nihil.

2) Description fonctionnelle:

Le sous-programme REGREP provoque l'impression de toutes les statistiques (calculées automatiquement par SIMUFOR) disponibles à propos de l'ensemble des régions de la simulation. Les données affichées sont divisées en deux grandes classes d'information qui ont été définies à la fin du chap. 3.

3) Position dans la chaîne des appels:

1. REGREP appelle le sous-programme ADDCAP.
2. Il peut être appelé par le programme principal, les sous-programmes de l'utilisateur ainsi que le sous-programme GENREP.

4) Description détaillée:

- Pour chaque classe d'information, le sous-programme REGREP parcourt la chaîne qui relie l'ensemble des régions "actives" au moment de l'appel, et imprime une ligne de résultats pour chacune d'entre elles.

- L'édition des résultats s'effectue, logiquement, pensons-nous, selon le même schéma que celui qui a été mis au point pour les stations simples et multiples.

4.11.7. Sous-programme INISTO (sto).

1) Argument:

sto: contient l'indice qui référence une station multiple.

2) Description fonctionnelle:

- INISTO réinitialise l'ensemble des variables statistiques (attributs-

système spécifiques) associées à la station multiple désignée par "sto".

- L'appel à cette procédure permet d'éliminer la partie non stationnaire qui est associée au démarrage d'un processus stochastique d'attente et de service. On peut ainsi espérer observer un processus stationnaire. Ceci validerait la pertinence des résultats que nous calculons. C'est surtout vrai en ce qui concerne les moyennes ergodiques et ce que nous avons appelé les "écarts-types ergodiques".

3) Position dans la chaîne des appels:

1. INISTO appelle parfois le sous-programme EERROR.

2. Il peut être appelé par le programme principal, les sous-programmes de l'utilisateur ainsi que le sous-programme GENINI.

4) Description détaillée:

- Si l'indice "sto" ne référence pas une station multiple, un message d'erreur (numéro 953) est affiché et la simulation est arrêtée.

- Les initialisations des variables statistiques sont sensiblement les mêmes que celles de la fonction NEWST. Seules les informations "courantes" restent inchangées et les informations "initiales" prennent les valeurs des informations "courantes".

4.11.8. Sous-programme INIFAC (fac).

1) Argument:

fac: contient l'indice qui référence une station simple.

2) Description fonctionnelle:

- INIFAC réinitialise l'ensemble des variables statistiques (attributs-système spécifiques) associées à la station simple désignée par "fac".

- L'utilité de cette procédure est la même que celle du sous-programme INISTO.

3) Position dans la chaîne des appels:

1. INIFAC appelle parfois le sous-programme EERROR.

2. Il peut être appelé par le programme principal, les sous-programmes de l'utilisateur ainsi que le sous-programme GENINI.

4) Description détaillée:

- Si l'indice "fac" ne référence pas une station simple, un message d'erreur (numéro 951) est affiché et la simulation est arrêtée.

- Les initialisations des variables statistiques sont sensiblement les mêmes que celles de la fonction NEWFAC. Comme dans le cas de INISTO, seules les informations "courantes" restent inchangées alors que les informations "initiales" prennent les valeurs des informations "courantes".

4.11.9. Sous-programme INIREG (reg).

1) Argument:

reg: contient l'indice qui référence une région.

2) Description fonctionnelle:

- INIREG réinitialise l'ensemble des variables statistiques (attributs-système spécifiques) associées à la région désignée par "reg".

- L'utilité de cette procédure n'est pas la même que celle de INISTO et INIFAC. En effet, INIREG permet simplement d'ajuster la période d'observation des statistiques relatives à la région désignée par "reg" sur celle d'autres entités déduites de GPSSS (stations simples et multiples principalement).

3) Position dans la chaîne des appels:

1. INIREG appelle parfois le sous-programme EERROR.

2. Il peut être appelé par le programme principal, les sous-programmes de l'utilisateur ainsi que le sous-programme GENINI.

4) Description détaillée:

- Si l'indice "reg" ne référence pas une région, un message d'erreur (numéro 950) est affiché et la simulation est arrêtée.

- Les initialisations des variables statistiques sont sensiblement les mêmes que celles de la fonction NEWREG. Seules les informations "courantes" restent inchangées et les informations "initiales" prennent les valeurs des informations "courantes".

4.11.10. Sous-programme GENINI.

1) Arguments: nihil.

2) Description fonctionnelle:

- Le sous-programme GENINI réinitialise l'ensemble des variables statistiques associées à toutes les stations simples et multiples ainsi qu'à toutes les régions de la simulation. L'appel à GENINI marque le démarrage d'une nouvelle période d'observation des statistiques automatiques de SIMUFOR.

- Remarquons que l'appel aux sous-programmes d'édition des statistiques ne provoque pas la réinitialisation de ces dernières, afin de permettre l'impression de résultats partiels sans troubler le processus de cumul des informations. L'utilisateur désireux d'obtenir plusieurs impressions à propos de périodes d'observation indépendantes doit employer les procédures de réinitialisation des calculs statistiques, qui sont à sa disposition.

3) Position dans la chaîne des appels:

1. GENINI appelle les sous-programmes INISTO, INIFAC et INIREG.

2. Il peut être appelé par le programme principal et les sous-programmes de l'utilisateur.

4) Description détaillée:

- On parcourt d'abord la chaîne des stations multiples afin de réinitialiser les variables statistiques de chacune d'entre elles, par l'appel à INISTO.

- On fait ensuite de même pour les stations simples et enfin, pour les régions.

4.11.11. Sous-programme GENREP.

1) Arguments: nihil.

2) Description fonctionnelle:

Le sous-programme GENREP provoque l'impression de toutes les statistiques automatiques calculées par SIMUFOR. Ce rapport couvre les résultats à propos des entités "GPSSS" suivantes:

* les stations multiples;

* les stations simples;

* les régions;

* les histogrammes.

3) Position dans la chaîne des appels:

1. GENREP appelle les sous-programmes STOREP, FACREP, REGREP et HISREP.

2. Il peut être appelé par le programme principal et les sous-programmes de l'utilisateur.

4) Description détaillée:

Après l'impression d'une page de garde, les résultats sont affichés par des appels successifs aux sous-programmes STOREP, FACREP, REGREP et HISREP.

4.12. SOUS-PROGRAMMES ET FONCTIONS ASSEMBLEUR.

Avec les fonctions et les sous-programmes écrits en ASSEMBLEUR, nous entrons dans le domaine de ce qui touche de très près à la portabilité de notre système SIMUFOR. Si nous implantons tous les objets de la simulation dans un grand commun blanc, toutes les procédures peuvent être implémentées en FORTRAN. Leurs caractéristiques de portabilité sont donc celles d'un langage comme le FORTRAN, très répandu et bien normalisé.

Nous nous sommes efforcés de n'utiliser que des instructions prévues explicitement dans la norme de FORTRAN. De cette façon, nous sommes persuadés que la réécriture de SIMUFOR, pour son utilisation sur un autre type d'ordinateur que le DEC 2060, ne prendrait qu'un minimum de temps.

Seules quatre procédures ont dû être écrites en langage d'assemblage. Celles-ci devront fatalement être totalement revues dans un nouvel environnement[3]. Il s'agit de la fonction LOCF ainsi que des sous-programmes SISAVE, JUMPTO et NET que nous nous proposons de décrire dans les pages qui suivent. Cependant, quelques mises au point doivent être faites au préalable.

En FORTRAN, l'appel du sous-programme SSPRGM par l'ordre "call SSPRGM" provoque d'abord le sauvetage de "l'adresse de retour dans le programme appelant". Il s'agit de l'adresse de l'instruction qui suit immédiatement l'ordre "call SSPRGM" dans le programme appelant. Il y a alors branchement au point d'entrée de SSPRGM.

L'instruction RETURN du sous-programme SSPRGM provoque, elle, la poursuite de l'exécution du programme appelant à l'adresse sauvée lors de l'exécution de l'ordre CALL.

C'est le mécanisme classique de l'exécution d'un sous-programme FORTRAN. Ce mécanisme est insuffisant pour réaliser la gestion des processus de simulation et l'implémentation de la notion de "co-routine". En effet, la suspension et la

---

[3]Le langage d'assemblage peut être fondamentalement différent d'un ordinateur à un autre.

reprise de l'exécution d'un processus, telles qu'elles ont été définies dans le chapitre 3, nécessitent l'écriture de deux sous-programmes ASSEMBLEUR:

1. le premier doit permettre de mémoriser, dans l'attribut-système LSC d'un processus suspendu, l'adresse à laquelle devra reprendre l'exécution de ce processus. Il s'agit du sous-programme SISAVE;

2. le second doit permettre de reprendre effectivement l'exécution d'un processus à l'adresse qui aura été sauvée par le sous-programme SISAVE précisément. Il s'agit du sous-programme JUMPTO.

Intuitivement, on pressentait que l'écriture de ces deux procédures ne devait poser aucun problème. L'expérience nous a montré que cet optimisme était quelque peu prématuré. En effet, la simplicité de l'écriture de ces procédures dépend de l'ordinateur avec lequel on travaille. A Montréal, nous avons utilisé un ordinateur CDC CYBER. Pour cet ordinateur, nous n'avons eu aucune peine à concevoir et à écrire rapidement ces deux procédures. Par contre, en ce qui concerne le DEC 2060, notre tâche s'est avérée beaucoup plus ardue. A quoi cela est-il dû?

Pour exécuter les ordres CALL et RETURN, l'implémenteur du langage FORTRAN doit sauver quelque part l'adresse de retour et trouver un moyen de réaliser les sauts (JUMP) nécessaires. Il existe deux grandes philosophies possibles en matière de sauvetage d'adresses et de sauts:

1. la première consiste, lors de l'appel d'un sous-programme (ordre CALL), à sauver l'adresse de retour dans le premier mot-mémoire du sous-programme appelé. Le retour au programme appelant (ordre RETURN) consiste alors à exécuter un saut inconditionnel à l'adresse mémorisée dans le premier mot-mémoire du sous-programme;

2. la seconde consiste, lors de l'appel d'un sous-programme (ordre CALL), à sauver l'adresse de retour dans une pile ("stack" global). Le retour au programme appelant consiste alors à retirer l'adresse (mémorisée lors de l'appel) de la pile et d'y effectuer un branchement.

Control Data a choisi la première solution. Dans ce cas, lors de l'appel à un sous-programme de cédulation, le rôle de SISAVE se résume à aller chercher le contenu du premier mot-mémoire du sous-programme en question et de le sauver dans l'attribut-système LSC du processus courant. Quant au rôle de JUMPTO, il se resume à rechercher l'adresse mémorisée dans l'attribut-système LSC du

processus dont on désire reprendre l'exécution, et à effectuer un branchement inconditionnel à cette adresse. Comme nous le voyons, les deux procédures SISAVE et JUMPTO sont faciles à comprendre et à écrire dans un environnement CDC.

Chez Digital, le problème provient de l'utilisation d'une pile pour sauver les adresses de retour. L'exécution de chaque ordre CALL (ou de chaque appel de fonction) sauve une adresse de retour dans la pile. L'exécution d'un programme FORTRAN est comparable, au point de vue de l'usage des sous-programmes, à une expression parenthésée (un ordre CALL équivaudrait à une parenthèse ouvrante, et un ordre RETURN, à une parenthèse fermante). Comme pour la compilation des expressions, l'usage d'une pile paraît donc tout à fait indiqué. Chaque ordre RETURN sera mis en relation avec la "bonne" adresse de retour. Tout serait parfait si nous utilisions FORTRAN de façon standard. Ce n'est malheureusement pas le cas. Notre problème vient du fait que nous exécutons beaucoup plus d'ordres CALL que d'ordres RETURN. Le contenu de la pile croît donc très rapidement. Or, ce contenu est limité à une soixantaine d'adresses de retour. Passé ce stade, l'exécution du programme s'arrête, faute de place dans la pile. Il nous faudra donc nettoyer régulièrement la pile (c'est-à-dire enlever les adresses de retour inutiles), sans pour autant troubler les appels (et surtout les retours) classiques de sous-programmes, effectués selon la norme de FORTRAN.

La figure 4-5 indique le schéma de l'appel et de l'exécution de tout sous-programme de gestion des processus (sauf ENDPRO). Seuls les appels (CALL) auxquels ne correspond aucun retour (RETURN) ont été indiqués. Il s'agit de l'appel au sous-programme de cédulation lui-même (call (1)), de l'appel à RESUME (call (2)) et de l'appel à JUMPTO (call (3)). Parmi les appels qui seront suivis d'un retour classique, seul l'appel à SISAVE a été mentionné, car ce sous-programme modifiera le contenu de la pile. En regard de la figure 4-5, nous indiquons l'évolution du contenu de la pile, tout au long de la séquence d'instructions réprésentée.

Nous remarquons que, à la fin de la séquence d'exécution présentée (qui correspond, rappelons-le, à l'exécution de toute procédure de cédulation, y compris les sous-programmes ENTFAC, ENTSTO et JOIN), la pile s'est enrichie de

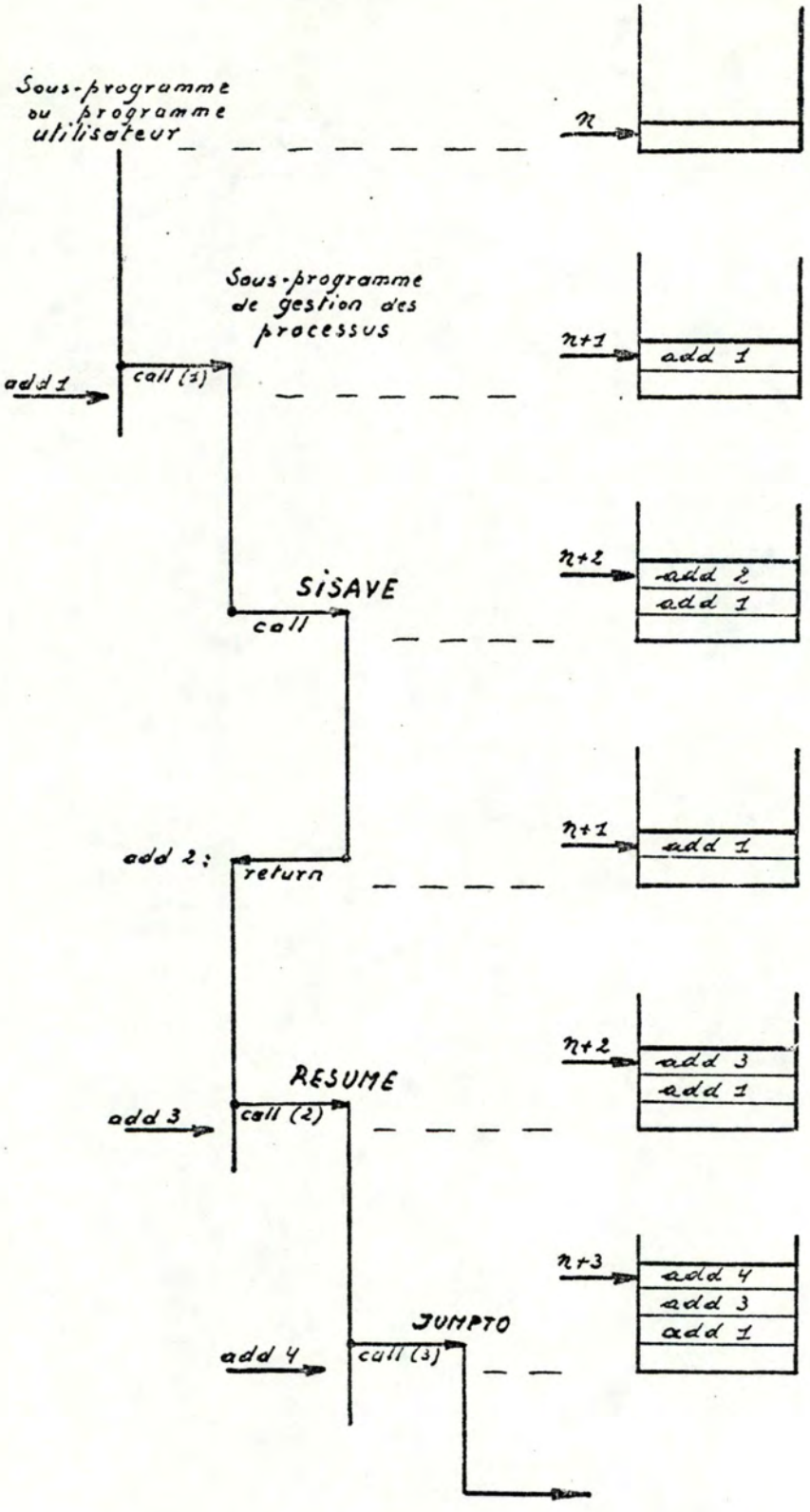


Figure 4-5: Exécution d'une procédure de cédulation.

trois adresses de retour (add1, add2, et add3) qui, non seulement ne seront jamais utilisées, mais surtout, ne seront jamais retirées. Les sous-programmes SISAVE et JUMPTO devront donc se charger, en plus de leur tâche originelle (qui est grosso modo la même que dans le cas du CDC), de nettoyer la pile des adresses de retour. On voit clairement que SISAVE peut enlever add1 de la pile, alors que JUMPTO peut, quant à lui, enlever les deux adresses add3 et add4. De cette façon, toutes les adresses de retour parasites seront éliminées de la pile.

Nous renvoyons le Lecteur désireux de s'informer plus amplement sur le passage des paramètres entre un sous-programme FORTRAN et un sous-programme ASSEMBLEUR aux manuels FORTRAN [8] et MACRO-20 [13] (ASSEMBLEUR) de Digital. Il trouvera là, également, la signification exacte des ordres POP, PUSH, POPJ et PUSHJ de manipulation de pile (éventuellement combinée à des branchements, dans le cas de POPJ et de PUSHJ).

A l'aide de la figure 4-5 toujours, nous pouvons à présent décrire les sous-programmes SISAVE et JUMPTO.

4.12.1. Sous-programme assembleur système SISAVE (var).

1) Argument:

var: désigne la variable dans laquelle on désire mémoriser l'adresse de retour.

2) Description fonctionnelle:

Le sous-programme SISAVE sauvera, dans le paramètre actuel de l'appel, l'adresse à laquelle reprendra, par la suite, l'exécution du processus que l'on est en train de suspendre par l'appel à un sous-programme de cédulation.

3) Position dans la chaîne des appels:

SISAVE est appelé par les sous-programmes ACTIV, PACTIV, REACT, PREACT, PASSIV, HOLD, WAIT, CANCEL, ENDPRO, ENTST et ENTFAC.

4) Description détaillée:

- Une pile n'est accessible que par son sommet. Le sous-programme SISAVE enlève donc les deux adresses qui se trouvent au sommet de la pile. Il renvoie la seconde adresse retirée (add1) dans le premier et seul argument actuel de l'appel.

- L'adresse add1 n'est pas replacée sur la pile. Le premier nettoyage du "stack global" a donc lieu ici.

- Par contre, la seconde adresse de retour (add2) est replacée au sommet de la pile. L'instruction POPJ réalise le mécanisme de retour classique (chez DEC) d'un sous-programme appelé. Elle enlève l'adresse de retour (add2) de la pile et effectue un branchement inconditionnel à cette adresse.

4.12.2. Sous-programme assembleur système JUMPTO (var).

1) Argument:

var: désigne la variable dans laquelle se trouve l'adresse vers laquelle ont désire effectuer un branchement.

2) Description fonctionnelle:

Le sous-programme JUMPTO effectue un branchement inconditionnel à l'adresse contenue dans le paramètre actuel de l'appel. Il permet de reprendre l'exécution d'un processus suspendu, à l'endroit où elle avait été interrompue, pourvu que l'on ait sauvé au préalable l'adresse de retour correspondante à l'aide du sous-programme SISAVE.

3) Position dans la chaîne des appels:

JUMPTO est appelé par le sous-programme RESUME.

4) Description détaillée:

- JUMPTO enlève les deux adresses qui se trouvent au sommet de la pile. Ce sont les adresses add3 et add4 qui ne sont d'aucune utilité et qui sont ainsi éliminées à leur tour (après add1). C'est donc ici que se produit le second nettoyage de la pile.

- Ensuite, JUMPTO met au sommet de la pile le contenu du premier (et seul) argument actuel.

- La poursuite de l'exécution du processus suspendu s'enclenche alors par l'ordre classique POPJ.

4.12.3. Sous-programme assembleur système NET.

1) Arguments: nihil

2) Description fonctionnnelle:

- Le seul but de ce sous-programme est de retirer l'adresse de retour qui se trouvait au sommet de la pile juste avant son appel.

- Pourquoi ce sous-programme? Nous avons démontré que l'appel à SISAVE puis à JUMPTO dans tous les sous-programmes de gestion des processus nettoyait totalement la pile des adresses de retour parasites. Il nous restait à envisager la cas de ENDPRO. Celui-ci marque la fin d'un processus. Par définition, aucune adresse de retour ne doit être sauvée pour un processus auquel on ne reviendra jamais. L'appel à SISAVE est donc superflu dans ENDPRO. Malheureusement, renoncer à SISAVE correspond à laisser l'adresse de retour parasite addl dans la pile, lors de la fin de chaque processus. Après la destruction d'une soixantaine de processus, c'est à nouveau l' engorgement de la pile. Ce dernier ne peut être évité que si l'on remplace, dans ENDPRO, l'appel à SISAVE par l'appel à NET dont le code est très proche de SISAVE. La pile est nettoyée de la même façon (élimination de addl); seule l'adresse de retour dans le processus n'est pas sauvée.

3) Position dans la chaîne des appels:

NET est appelé par le sous-programme ENDPRO.

4.12.4. Fonction assembleur LOCF (var).

1) Argument:

var: désigne une variable locale ou bien le point d'entrée d'un sous-programme.

2) Description fonctionnelle:

- Cette fonction permet, pour l'utilisateur, deux applications fort importantes.

  1. Elle renvoie l'adresse d'implantation d'une variable locale dont on donne le nom en argument (cfr. (1)).

  2. Elle renvoie également l'adresse du point d'entrée d'un sous-programme dont on donne le nom en argument. Pour cette seconde utilisation, le nom du sous-programme doit être déclaré par un ordre EXTERNAL (cfr. (2)).

```
      PROGRAM PRGM
      .
      .
      INTEGER CLASS
      EXTERNAL TOTO
      .
      .
(1)   ADD1 = LOCF (CLASS)
      .
(2)   ADD2 = LOCF (TOTO)
      .
      .
      STOP
      .
      .
      SUBROUTINE TOTO
      .
      .
      .
      RETURN
```

- LOCF est le seul sous-programme assembleur à être accessible à l'utilisateur. Ce dernier en aura besoin pour déterminer l'adresse du début du code associé à une classe de processus. Il devra passer cette adresse comme argument lors de l'appel à la fonction CLASS qui déclare la classe du processus en question.

3) Position dans la chaîne des appels:

LOCF peut être appelée par le programme principal et les sous-programmes de l'utilisateur; elle est appelée, dans tous les cas, par le sous-programme INIT.

CHAPITRE 5

CONCLUSIONS

5. CONCLUSIONS.

5.1. ETAPES DU TRAVAIL.

Arrivés au terme de ce travail, nous croyons utile de faire une brève rétrospective des étapes que nous avons suivies tout au long de l'élaboration de SIMUFOR.

5.1.1. Les deux versions de SIMUFOR.

Au cours des premiers mois pendant lesquels nous avons travaillé sous la direction du Prof. VAUCHER, nous avons tenté de mettre au point ce que nous appellerons ici la première version de SIMUFOR bien qu'elle en soit davantage une première approche. Malheureusement, pour les raisons que nous expliquons ci-après (cf. par. 5.1.2), cette tentative ne fut pas couronnée du succès que nous espérions; grâce à l'aide du Prof. VAUCHER, nous avons cependant pu modifier la conception même du travail, tout en reprenant les idées fondamentales de la première version. Nous avons ainsi abouti à un noyau de base que nous avons pu faire "tourner" sur le CDC avant de quitter le Canada.

Depuis notre retour en Belgique, nous avons poursuivi l'élaboration de la version de SIMUFOR présentée dans le présent travail et l'avons adaptée aux caractéristiques du DEC des Facultés Universitaires de Namur, sur lequel elle est opérationnelle.

Ces travaux sur l'ordinateur CDC puis sur le DEC nous ont permis de nous familiariser avec les caractéristiques de chacun d'eux et surtout de mesurer les difficultés du problème de portabilité: si le FORTRAN possédait tous les outils nécessaires à la réalisation d'un travail comme le nôtre, SIMUFOR aurait pu être utilisé, tant sur le DEC que sur le CDC, sans changement aucun dans le code des sous-programmes de la bibliothèque. Puisque, à Montréal, nous avons été amenés à faire appel en partie au langage d'assemblage COMPASS, nous avons été contraints, à Namur, de réécrire certains sous-programmes en ASSEMBLEUR. Ceci nous a fait comprendre combien il est malaisé d'adopter, pour les utiliser sur d'autres ordinateurs, des programmes écrits en langage d'assemblage. C'est pour ces raisons que nous estimons utile, dans la résolution d'un problème comme celui qui nous a été proposé, de limiter autant que faire se peut l'utilisation

de langages d'assemblage, c'est-à-dire de ne pas dépasser, dans leur emploi, les limites réellement indispensables.

5.1.2. Différences entre les deux versions.

Comme dit plus haut, la version finale de SIMUFOR s'écarte dans une certaine mesure de la première. En effet, au départ, nous avons voulu incorporer dans le système des éléments qui se sont avérés par la suite trop sophistiqués, en ce sens qu'ils conduisaient à des difficultés au "debugging" et à un ralentissement de l'exécution.

Nous avons alors adopté la stratégie du Prof. VAUCHER: nous sommes, avec lui, partis d'un noyau de base puis avons tenté de l'améliorer.

Les différences essentielles entre la version finale ici présentée et la première approche s'observent aux niveaux suivants:

1. La première version cherchait à augmenter la protection et donc la transparence de certaines variables-système; il existait, en effet, deux types d'indices:

   - ceux propres au système et par conséquent inaccessibles à l'utilisateur, et
   - ceux spécialement destinés à l'utilisateur.

   Il est apparu qu'il était nécessaire de passer constamment d'un indice à l'autre ce qui, par conséquent, ralentissait fortement l'exécution.
   La seconde version ne contient plus qu'un type d'indices, (cf. par. 3.2.3) commun au système et à l'utilisateur: l'exécution est donc accélérée mais au détriment de la protection des variables-système.

2. Le descripteur de classe de la première version était créé de manière dynamique, ce qui permettait de ne pas limiter le nombre maximum de classes. Par souci de simplicité et d'uniformité au niveau des éléments de l'espace d'adressage, nous avons renoncé à cette recherche de dynamisme, d'autant plus que la version finale présentée permet, sans trop de difficultés, d'augmenter, si le besoin s'en fait sentir, ce nombre maximum de classes.

3. Enfin, dans la version finale, nous avons préféré fournir à l'utilisateur des outils peut-être moins généraux et de finesse plus limitée au niveau de l'insertion des objets dans les listes, mais qui se rapprochent davantage de ceux qui constituent la caractéristique de SIMULA.

5.2. AVANTAGES ET INCONVENIENTS DE SIMUFOR.

5.2.1. Avantages.

Nous estimons que le système proposé présente, par rapport à SIMULA et à d'autres langages de simulation, les avantages suivants:

- SIMUFOR est très proche des primitives essentielles de SIMULA (cf. par. 5.1.2.3), tout en bénéficiant de la forte diffusion de FORTRAN et par conséquent de la possibilité d'une plus large utilisation potentielle dans le monde des utilisateurs des ordinateurs.

- Comme déjà dit, aux avantages de SIMULA sont ajoutés ceux de GPSS, ce qui donne à SIMUFOR une aptitude accrue à traiter certains problèmes de simulation et notamment ceux basés sur les files d'attente.

- Grâce à la simplicité de son code, SIMUFOR est plus aisément modifiable par l'utilisateur averti; celui-ci pourrait d'ailleurs implémenter les quelques prolongements que nous citons plus loin.

5.2.2. Inconvénients.

Il nous paraît naturel que SIMUFOR soit encore imparfait à ce stade de nos travaux; d'ailleurs la plupart des limitations de SIMUFOR sont, pour nous, liées à la nature même du FORTRAN. Parmi ces limitations, nous relevons ce qui suit:

- Il ne nous a pas été possible, sans compliquer exagérément notre système, de créer les concepts de sous-classes et de "ramasseur de miettes" ("garbage collector") qui existent dans SIMULA et en constituent un des avantages.

- De même, la façon la plus pratique d'obtenir, en FORTRAN, des variables globales à tous les sous-programmes et fonctions est l'utilisation de "commons"; l'utilisateur de SIMUFOR devra donc introduire, dans des "commons", toutes les variables dont il a besoin, ce qui représente un travail certes fastidieux mais combien nécessaire!

- Le Lecteur comprendra (cf. par. 2.8.2) que nous n'avons pas implémenté des outils qui facilitent le traitement de textes; ceux-ci sont présents dans SIMULA mais ne constituent pas un élément de base d'une simulation. L'utilisateur devra donc se contenter de la pauvreté relative de FORTRAN à ce niveau. Le traitement de texte en FORTRAN représenterait d'ailleurs, et à lui seul, l'objet d'un travail de longue haleine.

- La version actuelle de SIMUFOR offre moins de protection (c'est-à-dire moins de transparence) vis-à-vis de certaines variables essentielles à la simulation. Cependant, cet inconvénient nous paraît être compensé, dans une certaine mesure au moins, par une plus grande rapidité d'exécution (cf. par. 5.1.2.1).

5.3. LISTE DES POINTS DELICATS DE SIMUFOR.

Nous croyons utile de passer en revue, ne serait-ce que rapidement, les principaux points délicats de SIMUFOR, c'est-à-dire ceux qui requièrent une attention particulière de l'utilisateur.

1. L'utilisation de SIMUFOR exige, au titre de condition sine qua non, que ne soient pas modifiés les indices permettant de référencer les objets de la simulation et qui sont fournis par les fonctions que nous avons créées.

2. Etant donné que l'espace mémoire disponible est limité, l'utilisateur ne doit pas oublier, en fin de chaque sous-programme décrivant ses processus, de faire appel à ENDPRO; celui-ci lui permet de libérer et donc de récupérer l'espace mémoire dont il n'a plus besoin.

3. Nous pensons pouvoir conseiller à l'éventuel utilisateur de SIMUFOR de ne pas profiter de la déclaration implicite fournie par FORTRAN et donc de déclarer toutes les variables et toutes les fonctions qui lui sont utiles.

4. Sous sa forme actuelle, SIMUFOR ne permet pas le recouvrement des stations simples, des stations multiples et des régions sous peine d'incohérence statistique (cf. par. 3.8.11).

5. Enfin, il semble évident que l'on ne peut appeler les sous-programmes de cédulation ainsi que les sous-programmes qui concernent les éléments fondamentaux de GPSS (stations simples, stations multiples, régions, groupes) qu'à partir de sous-programmes décrivant des processus.

5.4. PROLONGEMENTS DE SIMUFOR.

Il va de soi que SIMUFOR n'aborde pas de façon exhaustive tous les aspects du problème qui nous a été posé. En conséquence, nous croyons utile de suggérer un certain nombre de voies qu'il serait intéressant ou nécessaire de suivre si notre SIMUFOR était jugé suffisamment digne d'intérêt pour que soit envisagée la possibilité de lui apporter des modifications ou des extensions. C'est ce que nous entendons par le terme "prolongements".

Voici quelques-unes de ces suggestions basées sur les limitations du présent travail et mises en évidence principalement aux par. 3.7.10 et surtout 3.8.11. Il serait utile de pouvoir:

- implémenter l'algorithme du WAIT UNTIL;
- introduire des statistiques plus sophistiquées;
- améliorer l'organisation de l'échéancier (SOS);
- permettre à un processus de figurer simultanément sur plusieurs listes;
- autoriser le recouvrement dans l'utilisation de stations simples, stations multiples et régions;
- poursuivre l'étude (amorcée par nos soins mais sans succès particulier suite à un manque réel de documentation du constructeur DEC) de l'implantation dynamique des objets de la simulation;
- réaliser la "préemption" pour les stations simples et multiples.

Remarques:

1. On pourrait encore suggérer d'effectuer des mesures de performances de SIMUFOR et de les comparer avec celles de SIMULA, notamment en ce qui concerne les temps d'exécution des programmes. Remarquons néanmoins que SIMULA est un compilateur alors que SIMUFOR est une bibliothèque de sous-programmes écrits en FORTRAN, et par conséquent, qu'une comparaison des performances présente des difficultés réelles.
2. On pourrait enfin suggérer la réalisation d'un précompilateur qui permettrait de supprimer dans une mesure plus ou moins large, les contraintes inhérentes au FORTRAN. L'utilisateur pourrait alors disposer d'une notation par point (disponible en SIMULA), d'un plus grand nombre de caractères pour nommer une variable, d'une génération automatique des communs, etc...

BIBLIOGRAPHIE

[1] J. AGARD, J. ALTABER, R. FORTET, A. KAUFMANN, P. LE GALL, M. PRECIGOUT, G. THOMAS.
Les Méthodes de Simulation.
Dunod, Paris, 1968.

[2] M. BADEL et M. VERAN.
FORTSIM: un système d'aide à l'écriture de simulateurs en FORTRAN.
RAIRO Informatique Computer Science, 1979.

[3] J. BUREAU.
Dictionnaire de l'informatique.
Larousse, Paris, 1972.

[4] D.Y. DOWNHAM, R.D.K. ROBERTS.
Multiplicative Congruential Pseudo Random Number Generators.
Computer Journal 10, 1967.

[5] M. DREYFUS.
Fortran IV.
Dunod, Paris, 1973.

[6] J. FICHEFET.
La simulation.
FNDP, 1976.

[7] G.S. FISHMAN.
Concepts and Methods in Discrete Event Digital Simulation.
John Wiley and Sons, New York, London, Sidney, Toronto, 1973.

[8] Digital Equipment Corporation.
FORTRAN Reference Manual.
Digital Equipment Corporation, Maynard, Massachusetts, 1977.

[9] W.R. FRANTA.
The process view of simulation.
Elsevier, 1978.

[10] M.G. HARTLEY (editor).
Digital Simulation Methods.
Peregrinus Ltd, Southgate House, Stevenage, Herts SG1 1HQ, England, 1975.

[11] LAROUSSE (editor).
Larousse trois volumes.
LAROUSSE, 1966.

[12] J. LEROUDIER, D. RENAULT, M. RENAULT.
Etude de la gestion des événements dans une simulation à événements discrets.
IRIA Laboria, 1977.

[13] Digital Equipment Corporation.
MACRO20 Reference Manual.
Digital Equipment Corporation, Maynard, Massachusetts, 1977.

[14] T.H. NAYLOR, J.L. BALINTFY, D.S. BURDICK, KONG CHU.
Computer Simulation Techniques.
John Wiley et Sons, Inc. ,New-York, London, Sidney , 1968.

[15] M. NOIRHOMME-FRAITURE.
SIMULA notes personnelles prises lors des séminaires donnés à la FNDP.
1980.

[16] J. SMITH.
Computer Simulation Models.
Griffin, London, 1968.

[17] J.G VAUCHER and P. DUVAL.
A Comparison of Simulation Event List Algorithms.
CACM, 1975.

[18] J.G. VAUCHER et P. BRATLEY.
La Simulation avec SIMULA.
Université de Montréal, 1979.

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR
Institut d'Informatique

ANNÉE ACADÉMIQUE 1980-1981

UN SYSTÈME D'AIDE À LA SIMULATION À ÉVÉNEMENTS DISCRETS EN FORTRAN

Annexe

Jean-Mathieu NISEN
et
José ROULIN

Mémoire présenté
en vue de l'obtention du grade de
Licencié et Maître
en Informatique

A N N E X E

1. SOUS-PROGRAMMES ET FONCTIONS FORTRAN.

```
c=============================================================
c
c
c
c  ROUTINES D'ORDRE GENERAL
c
c
c
c=============================================================
      subroutine init
c
c   initialisation des variables de simulation
c     navail = espace disponible dans zone dynamique
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      equivalence (evtime,lsc)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
c
c   allocation pseudo-dynamique de memoire en utilisant
c       un common blanc de longueur confortable
c       MAIS statique !!!
c
      common zad(50000)
      navail = 50000
      objlib = locf(zad) - locf(class) + 1
      maxobj = objlib + navail - 1
c
c   init des variables pour decrire les classes
c
      none = objlib
      maxcl = 20
      do 100 i=1,maxcl
         size(i) = 7
         old(i) = none
         lsc0(i) = 0
  100    continue
      head = 2
      notice = 3
      maint = 4
      storag = 5
      facily = 6
      groupe = 7
      region = 8
      histo = 9
      freecl = 10
c
      size (storag) = 44
      size (facily) = 32
```

```
      size (region) = 24
      size (histo) = 146
c
c   init de l'objet none avec attributs pour simplifier les
c   tests a l'execution
c
      none = new(1)
c
c   init du chainage des entites GPSSS
c
      laststst = none
      lastfc = none
      lastgr = none
      lastrg = none
      lasths = none
c
c   init de la sqs
c
      time = 0.0
      sqs = new(head)
      evtime(sqs) = -1.0
      inot = new(notice)
      actuel = new(maint)
      call into(inot,sqs)
      pt(inot) = actuel
      evtime(inot) = 0.0
      pt(actuel) = inot
c
c   la variable de systeme -main- represente le programme
c   principal
c
      main = actuel
c
c   init des codes d'activation
c
      after = 1
      before = 2
      delay = 3
      at = 4
c
c   par defaut, il n'y aura pas de tracage de la simulation
c
      itrace = 0
c
c   init des devices d'input et d'output
c
      input = 5
      output = 5
      call lien
c
  999 continue
      return
      end
c*************************************************************
      integer function class (loc,taille)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
```

```
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
c
      if (freecl .le. maxcl) go to 100
         call eerror(101)
         class = 1
         go to 999
  100 continue
      if (taille .ge. 7) go to 150
         call eerror(102)
         go to 999
  150 continue
      class = freecl
      size(freecl) = taille
      old(freecl) = none
c
c   calcul d'adresse pour l'utilisation de l'instruction PUSHJ
c
      lsc0(freecl)="310000000000 + loc
      freecl = freecl + 1
c
  999 continue
      return
      end
c*****************************************************************
      integer function new (klass)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      real ptstat
c
      if (itrace .gt. 0) write (output,10),klass
   10 format (1x,'-- new ',i5)
      if (old(klass) .eq. none) go to 100
         i = old(klass)
         old(klass) = suc(i)
         go to 300
  100 continue
      if (objlib+size(klass) .le. maxobj) go to 200
          call eerror(103)
          new = none
          go to 999
  200 continue
      i = objlib
      objlib = objlib + size(klass)
  300 continue
      new = i
      class(i) = klass
      suc(i) = none
      pred(i) = none
      pt(i) = none
      lsc(i) = lsc0(klass)
      prior(i) = 0
      ptstat(i) = 0.0
```

```
      if (klass .ne. head) go to 400
         suc(i) = i
         pred(i) = i
  400 continue
c
  999 continue
      return
      end
c*************************************************************
      subroutine return (obj)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
c
      if (obj .ne. none) go to 100
        call eerror(104)
        go to 999
  100 continue
        call sout(obj)
        suc(obj) = old(class(obj))
        old(class(obj)) = obj
        obj = none
c
  999 continue
      return
      end
c*************************************************************
      subroutine kill (obj)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),list(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
      dimension pt(1)
      equivalence (pt,list)
c
      logical empty
c
      if (obj .ne. main) go to 100
         call eerror (110)
         go to 999
c
  100 continue
      if ( .not. ((( class(obj) .eq. head   ) .or.
     1            ( class(obj) .eq. storag ) .or.
     2            ( class(obj) .eq. facily ) .or.
     3            ( class(obj) .eq. groupe )
     4           ) .and.     ( .not. empty(obj))
     5          )
     6    ) go to 200
           call werror (101)
           go to 999
```

```
c
c liste = storage
c
  200 continue
      if ( class(obj) .ne. storag ) go to 220
         if ( obj .ne. laststt ) go to 210
              laststt = list (obj)
              go to 800
c
  210 continue
      ljml = laststt
      lj = list (laststt)
      go to 600
c
c liste = facily
c

  220 continue
      if ( class(obj) .ne. facily ) go to 240
         if ( obj .ne. lastfc ) go to 230
              lastfc = list (obj)
              go to 800
c
  230 continue
      ljml = lastfc
      lj = list (lastfc)
      go to 600
c
c liste = groupe
c
  240 continue
      if ( class(obj) .ne. groupe ) go to 260
         if ( obj .ne. lastgr ) go to 250
              lastgr = list (obj)
              go to 800
c
  250 continue
      ljml = lastgr
      lj = list (lastgr)
      go to 600
c
c liste = region
c
  260 continue
      if ( class(obj) .ne. region ) go to 280
         if ( obj .ne. lastrg ) go to 270
              lastrg = list (obj)
              go to 800
c
  270 continue
      ljml = lastrg
      lj = list (lastrg)
      go to 600
c
c liste = histo
c
  280 continue
      if ( class(obj) .ne. histo ) go to 500
         if ( obj .ne. lasths ) go to 290
```

```
            lasths = list (obj)
            go to 800
c
  290 continue
      ljml = lasths
      lj = list (lasths)
      go to 600
c
  500 continue
      go to 800
c
  600 continue
      if ( lj .eq. obj ) go to 650
          ljml = lj
          lj = list (lj)
          go to 600
c
  650 continue
      list (ljml) = list (lj)
      go to 800
c
  800 continue
      call return (obj)
      go to 999
c
  999 continue
      return
      end
c*****************************************************************
      subroutine lien
c
      implicit integer (a-z)
      common /io/ input,output
c
      write (output,10)
      write (output,11)
      write (output,12)
      write (output,13)
      write (output,14)
c
   10 format (' ----------------------------------------')
   11 format (' '                                        )
   12 format ('                SIMUFOR                   ')
   13 format (' '                                        )
   14 format (' ----------------------------------------')
c
    5 continue
c
      write (output,100)
      write (output,101)
      write (output,102)
      write (output,103)
      write (output,104)
      write (output,105)
      write (output,106)
      write (output,107)
      write (output,108)
c
  100 format (' Pour l''impression de vos resultats, vous avez ')
```

```
  101 format (' le choix entre : ')
  102 format ('    -l''imprimante (frappez 3) ')
  103 format ('    -votre terminal (frappez 5) ')
  104 format ('    -un fichier sur disque (frappez un nombre xx ')
  105 format ('                    compris entre 16 et 63 : vos ')
  106 format ('                    resultats se trouveront alors')
  107 format ('                    dans un fichier FORxx.DAT.)')
  108 format (' Votre chiffre d''output : ')
      read (input,200),x
  200 format (i3)
c
      if ( x .eq. 3 .or. x .eq. 5 .or. (x .ge. 16 .and. x .lt. 63))
     1    goto 300
      go to 5
c
  300 continue
      output = x
c
  999 continue
      return
      end
c*************************************************************
      subroutine temps (vartps)
c
      real time,vartps
      common /tmps/ time
c
      vartps = time
c
  999 continue
      return
      end
c*************************************************************
      subroutine setpr (obj,val)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (class(obj) .gt. histo .or. obj .eq. main) prior(obj) = val
c
  999 continue
      return
      end
c*************************************************************
      logical function idle (proc)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /gpsss/ storag,facily,groupe,region,histo
c
      if ( class(proc) .gt. histo .or. proc .eq. main) go to 100
           call eerror (105)
           go to 999
  100 continue
```

```
      idle = pt(proc) .eq. none
c
  999 continue
      return
      end
c*************************************************************
      real function evtime (proc)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /gpsss/ storag,facily,groupe,region,histo
      logical idle
      real evt(1)
      equivalence (evt,lsc)
c
      if ( class(proc) .gt. histo .or. proc .eq. main ) go to 100
           call eerror (106)
           go to 999
  100 continue
      if (.not. idle(proc)) go to 200
         call eerror (107)
         go to 999
  200 continue
      evtime = evt(pt(proc))
c
  999 continue
      return
      end
```

```
c=============================================================
c
c
c
c  GESTION DES PROCESSUS
c
c
c
c=============================================================
      subroutine resume
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      equivalence (evtime,lsc)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      logical empty
c
      if (empty(sqs)) go to 100
         actuel = pt(suc(sqs))
         time = evtime(suc(sqs))
         if (itrace .gt. 0) write (output,10),time
   10    format (1x,' time = ',f8.3)
         if (itrace .gt. 1) call dumps
         if (itrace .gt. 1) call suivre
         call jumpto (lsc(actuel))
         go to 999
  100 continue
      write (output,20)
   20 format (1x,'*** fin prematuree - sqs vide')
c
      stop
c
  999 continue
      return
      end
c**************************************************************
      subroutine activ (obj,code,dt)
c
      implicit integer (a-z)
      real dt,t
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      real bidon
      equivalence (bidon,dtt)
      equivalence (evtime,lsc)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      bidon = dt
      if (obj .ne. none) go to 100
         call werror(201)
```

```
          go to 999
  100 continue
      call sisave(lsc(actuel))
      if (pt(obj) .eq. none) go to 200
         call eerror(201)
         go to 999
c
  200 continue
      n = new(notice)
  300 continue
      go to (340,340,310,320) ,code
            call eerror(202)
            go to 999
c
c   code = delay
c
  310 continue
      t = time + dt
      if (t .ge. time) go to 312
         call werror(202)
         t = time
c
  312 continue
      evtime(n) = t
      i = pred(sqs)
  314 if (evtime(i) .le. t) go to 316
         i = pred(i)
         go to 314
c
  316 continue
      call sfolow (n,i)
      go to 940
c
c   code = at
c
  320 continue
      t = dt
      if (t .ge. time) go to 322
         call werror(203)
         t = time
c
  322 continue
      evtime(n) = t
      i =  pred(sqs)
  324 if (evtime(i) .le. t) go to 326
         i = pred(i)
         go to 324
c
  326 continue
      call sfolow (n,i)
      go to 940
c
c   code = after or before
c
  340 continue
      if (pt(dtt) .ne. none) go to 342
         call werror(204)
         go to 999
  342 continue
```

```
      i = pt(dtt)
      evtime(n) = evtime(i)
      if (code .eq. before) go to 348
c
c   code = after
c
  900 continue
      call sfolow(n,i)
      go to 940
c
c   code = before
c
  348 continue
      call sprece(n,i)
c
  940 continue
      pt(n) = obj
      pt(obj) = n
      if (itrace .gt. 0) write (output,10),t
   10 format (1x,'-- activ pour ',f8.3)
      call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine react (obj,code,dt)
c
      implicit integer (a-z)
      real dt,t
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      real bidon
      equivalence (bidon,dtt)
      equivalence (evtime,lsc)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      bidon = dt
      if (obj .ne. none) go to 100
         call werror(201)
         go to 999
  100 continue
      call sisave(lsc(actuel))
      if (pt(obj) .eq. none) go to 200
         call return(pt(obj))
         pt(obj) = none
c
  200 continue
      n = new(notice)
  300 continue
      go to (340,340,310,320) ,code
            call eerror(202)
            go to 999
c
c   code = delay
```

```
c
  310 continue
      t = time + dt
      if (t .ge. time) go to 312
         call werror(202)
         t = time
c
  312 continue
      evtime(n) = t
      i = pred(sqs)
  314 if (evtime(i) .le. t) go to 316
         i = pred(i)
         go to 314
c
  316 continue
      call sfolow (n,i)
      go to 940
c
c   code = at
c
  320 continue
      t = dt
      if (t .ge. time) go to 322
         call werror(203)
         t = time
c
  322 continue
      evtime(n) = t
      i =  pred(sqs)
  324 if (evtime(i) .le. t) go to 326
         i = pred(i)
         go to 324
c
  326 continue
      call sfolow (n,i)
      go to 940
c
c   code = after or before
c
  340 continue
      if (pt(dtt) .ne. none) go to 342
         call werror(204)
         go to 999
  342 continue
      i = pt(dtt)
      evtime(n) = evtime(i)
      if (code .eq. before) go to 348
c
c   code = after
c
  900 continue
      call sfolow(n,i)
      go to 940
c
c   code = before
c
  348 continue
      call sprece(n,i)
c
```

```
  940 continue
      pt(n) = obj
      pt(obj) = n
      if (itrace .gt. 0) write (output,10),t
   10 format (1x,'-- react pour ',f8.3)
      call resume
c
  999 continue
      return
      end
c*************************************************************
      subroutine pactiv (obj,code,dt)
c
      implicit integer (a-z)
      real dt,t
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      equivalence (evtime,lsc)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      if (obj .ne. none) go to 100
         call werror(205)
         go to 999
  100 continue
      if (code .eq. at .or. code .eq. delay ) go to 200
         call eerror(203)
         go to 999
  200 continue
      call sisave(lsc(actuel))
      if (pt(obj) .eq. none) go to 300
         call eerror(204)
         go to 999
  300 continue
      n=new(notice)
      if (code .eq. at) go to 400
c
c   code = delay
c
      t=time+dt
      if (t .ge. time) go to 420
         call werror(206)
         t=time
         go to 420
c
c   code = at
c
  400 continue
      t=dt
      if (t .ge. time) go to 420
         call werror(207)
         t=time
c
  420 continue
      evtime(n)=t
      pl=prior(obj)
```

```
      i=suc(sqs)
c
c
  430 continue
      if (evtime(i) .ge. t .or. i .eq. sqs) go to 440
         i=suc(i)
         go to 430
c
  440 continue
      if (evtime(i) .gt. t) go to 460
  450    continue
         if (prior(i) .lt. pl) go to 460
            i=suc(i)
            go to 450
c
  460 continue
      call sprece(n,i)
      pt(n)=obj
      pt(obj)=n
      if (itrace .gt. 0) write (output,10),t
   10 format (1x,'-- pactiv pour ',f8.3)
      call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine preact (obj,code,dt)
c
      implicit integer (a-z)
      real dt,t
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      equivalence (evtime,lsc)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      if (obj .ne. none) go to 100
         call werror(205)
         go to 999
  100 continue
      if (code .eq. at .or. code .eq. delay ) go to 200
         call eerror(203)
         go to 999
  200 continue
      call sisave(lsc(actuel))
      if (pt(obj) .eq. none) go to 300
          call return(pt(obj))
          pt(obj) = none
  300 continue
      n=new(notice)
      if (code .eq. at) go to 400
c
c   code = delay
c
      t=time+dt
```

```
      if (t .ge. time) go to 420
         call werror(206)
         t=time
         go to 420
c
c   code = at
c
  400 continue
      t=dt
      if (t .ge. time) go to 420
         call werror(207)
         t=time
c
  420 continue
      evtime(n)=t
      pl=prior(obj)
      i=suc(sqs)
c
c
  430 continue
      if (evtime(i) .ge. t .or. i .eq. sqs) go to 440
         i=suc(i)
         go to 430
c
  440 continue
      if (evtime(i) .gt. t) go to 460
  450    continue
         if (prior(i) .lt. pl) go to 460
            i=suc(i)
            go to 450
c
  460 continue
      call sprece(n,i)
      pt(n)=obj
      pt(obj)=n
      if (itrace .gt. 0) write (output,10),t
   10 format (1x,'-- preact pour ',f8.3)
      call resume
c
  999 continue
      return
      end
c*************************************************************
      subroutine hold (dt)
c
      implicit integer (a-z)
      real dt,t
c
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      equivalence (evtime,lsc)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      call sisave (lsc(actuel))
      t = time + dt
```

```
      if (t .ge. time) go to 50
         call werror(208)
         t=time
   50 continue
      n = suc(sqs)
      evtime(n) = t
      if (itrace .gt. 0) write (output,10),dt,t
   10 format (1x,'-- hold de ',f8.3,' jusqu a ',f8.3)
      i = pred(sqs)
  100 if (evtime(i) .le. t) go to 200
         i = pred(i)
         go to 100
  200 continue
      call sfolow(n,i)
      call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine passiv
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /io/ input,output
      common /secret/ sqs,objlib,maxobj,notice,itrace
      real evtime(1)
      equivalence (evtime,lsc)
c
      call sisave (lsc(actuel))
      call return (pt(actuel))
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- passiv')
      call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine cancel (obj)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /io/ input,output
      real evtime(1)
      equivalence (evtime,lsc)
c
      call sisave(lsc(actuel))
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- cancel')
      if (pt(obj) .eq. none) go to 100
          call return(pt(obj))
          go to 300
  100 call werror(209)
```

```
  300 call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine wait (q)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      call sisave (lsc(actuel))
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- wait')
c
      call return (pt(actuel))
      call into (actuel,q)
      call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine endpro
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- fin de process --')
      call return (pt(actuel))
      call return (actuel)
      call net
      call resume
  999 continue
      return
      end
```

```
c=============================================================
c
c
c
c   TRAITEMENT DE LISTES
c
c
c
c=============================================================
      subroutine sout (j)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      i = pred(j)
      k = suc(j)
      suc(i) = k
      pred(k) = i
c
  999 continue
      return
      end
c*************************************************************
      subroutine sprece (j,k)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      if (j .eq. k) go to 999
        if (suc(j) .ne. none) call sout(j)
        i = pred(k)
        pred(j) = i
        suc(j) = k
        pred(k) = j
        suc(i) = j
c
  999 continue
      return
      end
c*************************************************************
      subroutine sfolow (j,i)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      if (i .ne. j) call sprece(j,suc(i))
c
  999 continue
      return
      end
c*************************************************************
      subroutine out (i)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
```

```
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      if (class(i) .gt. head) go to 100
        call eerror(301)
        go to 999
  100 continue
      if (suc(i) .eq. none) go to 999
        call sout(i)
        suc(i) = none
        pred(i) = none
c
  999 continue
      return
      end
c*****************************************************************
      subroutine preced (j,k)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      if (class(j) .gt. head .and.
     1    class(k) .ge. head .and.
     2    suc(k)   .ne. none     ) go to 100
        call eerror(302)
        go to 999
  100 continue
      call sprece(j,k)
c
  999 continue
      return
      end
c*****************************************************************
      subroutine follow (j,i)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      if (i .ne. j) call preced(j,suc(i))
c
  999 continue
      return
      end
c*****************************************************************
      subroutine into (i,l)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (class(l) .eq. head .or.
     1    class(l) .eq. storag .or.
     2    class(l) .eq. facily .or.
```

```
     3    class(1) .eq. groupe     ) go to 100
          call eerror(303)
          go to 999
  100 continue
      call preced(i,1)
c
  999 continue
      return
      end
c*****************************************************************
      logical function empty (1)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (class(1) .eq. head .or.
     1    class(1) .eq. storag .or.
     2    class(1) .eq. facily .or.
     3    class(1) .eq. groupe     ) go to 100
         call eerror(304)
         empty = .true.
         go to 999
  100 continue
         empty = suc(1) .eq. 1
c
  999 continue
      return
      end
c*****************************************************************
      integer function first (1)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      logical empty
c
      first = none
      if (.not. empty(1)) first = suc(1)
c
  999 continue
      return
      end
c*****************************************************************
      integer function last (1)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
c
      logical empty
c
      last = none
```

```
      if (.not. empty(l)) last = pred(l)
c
  999 continue
      return
      end
c*****************************************************************
      integer function suc(i)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(l),pred(l),izsuc(l),pt(l),lsc(l),prior(l)
      common /objatr/ ptstat(l)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
c
      suc = izsuc(i)
      if (class(izsuc(i)) .le. groupe) suc = none
c
  999 continue
      return
      end
c*****************************************************************
      integer function pred(i)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(l),izpred(l),suc(l),pt(l),lsc(l),prior(l)
      common /objatr/ ptstat(l)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
c
      pred = izpred(i)
      if (class(izpred(i)) .le. groupe) pred = none
c
  999 continue
      return
      end
c*****************************************************************
      integer function cardi (l)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(l),pred(l),suc(l),pt(l),lsc(l),prior(l)
      common /objatr/ ptstat(l)
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (class(l) .eq. head .or.
     1    class(l) .eq. storag .or.
     2    class(l) .eq. facily .or.
     3    class(l) .eq. groupe     ) go to 100
           call eerror (310)
           cardi = 0
           go to 999
  100 continue
      nb = 0
      suiv = suc(l)
  200 continue
      if (suiv .eq. l) go to 300
            suiv = suc(suiv)
```

```
            nb = nb + 1
            go to 200
  300 continue
      cardi = nb
c
  999 continue
      return
      end
```

```
c=============================================================
c
c
c
c     STORAGES
c
c
c
c=============================================================
      subroutine verist (storag)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      common /temsto/ ask,client,strg
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
      real ptstat
      real avc
      dimension pt(1),ptstat(1),requi(1)
      equivalence (requi,pt),(list,requi)
      equivalence (ptname,ptstat)
c
      strg = storag
      client = first(strg)
   50 continue
      if (client .eq. none) go to 800
         if (cap(strg) - conten(strg) .lt. requi(client)) go to 200
            call addcap(avcpq(strg),stdcpq(strg),nbcpq(strg),
     1                  mincpq(strg),maxcpq(strg),curcpq(strg),
     2                  time-tpscpq(strg))
            tpscpq(strg) = time
            call out(client)
            call addflt(avwt(strg),stdwt(strg),nbwt(strg),
     1                  minwt(strg),maxwt(strg),time-ptstat(client))
            ptstat(client) = time
            curcpq(strg) = curcpq(strg) - 1
            ask = requi(client)
            pt(client) = none
            call activ (client,delay,0.0)
            call addcap(avcps(strg),stdcps(strg),nbcps(strg),
     1                  mincps(strg),maxcps(strg),conten(strg),
     2                  time-tpscps(strg))
            tpscps(strg) = time
            conten (strg) = conten(strg) + ask
  200 continue
      client = suc(client)
      go to 50
```

```
c
  800 continue
      if (conten(sto) .gt. 0) go to 900
         tpus(strg) = tpus(strg) + (time - lstpus(strg))
         lstpus(strg) = time
c
  900 continue
      return
      end
c*************************************************************
      subroutine printo (i,l)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
      real ptstat
      dimension ptstat(1),prior(1)
      equivalence (ptname,ptstat),(conten,prior)
c
      p = first(l)
      if (p .ne. none) go to 200
         call into (i,l)
         go to 900
  200 continue
      if ( .not. ((prior(p) .lt. prior(i))
     1            .or.
     2            (  (prior(i) .lt. 0 )
     3               .and.
     4                (prior(i) .eq. prior(p))
     5            )
     6           )
     7    ) go to 400
         call sprece(i,p)
         go to 900
c
  400 continue
      p = last(l)
      if (prior(i) .ge. 0) go to 600
  500    continue
         if (prior(i) .lt. prior(p)) go to 800
            p = pred(p)
            go to 500
c
  600 continue
      if (prior(i) .le. prior(p)) go to 800
         p = pred(p)
```

```
         go to 600
c
  800 continue
      call sfolow(i,p)
c
  900 continue
      call addcap(avcpq(1),stdcpq(1),nbcpq(1),mincpq(1),maxcpq(1),
     1            curcpq(1),time-tpscpq(1))
      tpscpq(1) = time
      curcpq(1) = curcpq(1) + 1
c
  999 continue
      return
      end
c*************************************************************
      integer function newst(nomst,capac)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststst,lastfc,lastgr,lastrg,lasths
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
      real ptstat
      dimension nomst(3),ptstat(1)
      equivalence (ptname,ptstat)
c
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- new storage')
      if (old(storag) .eq. none) go to 100
         i = old(storag)
         old(storag) = suc(i)
         go to 300
  100 continue
      if (objlib+size(storag) .le. maxobj) go to 200
         call eerror(401)
         newst = none
         go to 999
  200 continue
      i = objlib
      objlib = objlib + size(storag)
  300 continue
      newst = i
      class(i) = storag
```

```
      suc(i) = i
      pred(i) = i
      list(i) = laststst
      laststst = i
      cap(i) = capac
      conten(i) = 0
      do 350 j=1,3
          ptname(i,j) = nomst(j)
 350  continue
      nbent(i) = 0
      nulent(i) = 0
      minwt(i) = 100000.0
      maxwt(i) = 0.0
      avwt(i) = 0.0
      stdwt(i) = 0.0
      nbwt(i) = 0
      inicpq(i) = 0
      mincpq(i) = "777777777
      maxcpq(i) = 0
      avcpq(i) = 0.0
      stdcpq(i) = 0.0
      nbcpq(i) = 0.0
      tpscpq(i) = time
      curcpq(i) = 0
      minst(i) = 10000.0
      maxst(i) = 0.0
      avst(i) = 0.0
      stdst(i) = 0.0
      nbst(i) = 0
      inicps(i) = 0
      mincps(i) ="777777777
      maxcps(i) = 0
      avcps(i) = 0.0
      stdcps(i) = 0.0
      nbcps(i) = 0.0
      tpscps(i) = time
      minrqs(i) = "777777777
      maxrqs(i) = 0
      avrqs(i) = 0.0
      stdrqs(i) = 0.0
      nbrqs(i) = 0
      tpus(i) = 0.0
      tpini(i) = time
      lstpus(i) = time
c
  999 continue
      return
      end
c*****************************************************************
      subroutine entst (sto,requi)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
```

```
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
      real ptstat
      dimension pt(1),lsc(1),prior(1),ptstat(1)
      equivalence (list,pt),(cap,lsc),(conten,prior),(ptname,ptstat)
c
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- entst')
      if (class(sto) .eq. storag) go to 200
         call eerror(402)
         go to 999
  200 if (requi .le. cap(sto)) go to 300
         call eerror(403)
         go to 999
  300 continue
      nbent(sto) = nbent(sto) + 1
      call addint(avrqs(sto),stdrqs(sto),nbrqs(sto),
     1            minrqs(sto),maxrqs(sto),requi)
      ptstat(actuel) = time
      if (conten(sto)+requi .gt. cap(sto)) go to 400
         nulent(sto) = nulent(sto) + 1
         call addcap(avcps(sto),stdcps(sto),nbcps(sto),mincps(sto),
     1               maxcps(sto),conten(sto),time-tpscps(sto))
         tpscps(sto) = time
         if (conten(sto) .le. 0) lstpus(sto) = time
         conten(sto) = conten(sto) + requi
         go to 999
  400 continue
      call printo(actuel,sto)
      call sisave(lsc(actuel))
      call return(pt(actuel))
      pt(actuel) = requi
      call resume
c
  999 continue
      return
      end
c*************************************************************
      subroutine least (sto,rlease)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
```

```
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
      real ptstat
      dimension pt(1),lsc(1),prior(1),ptstat(1)
      equivalence (list,pt),(cap,lsc),(conten,prior),(ptname,ptstat)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- least')
      if (class(sto) .eq. storag) go to 200
         call eerror(404)
         go to 999
  200 if (rlease .le. cap(sto)) go to 300
         call eerror(405)
         go to 999
  300 continue
      call addcap(avcps(sto),stdcps(sto),nbcps(sto),mincps(sto),
     1            maxcps(sto),conten(sto),time-tpscps(sto))
      tpscps(sto) = time
      conten(sto) = conten(sto) - rlease
      if (conten(sto) .lt. 0) go to 400
         call addflt(avst(sto),stdst(sto),nbst(sto),minst(sto),
     1               maxst(sto),time-ptstat(actuel))
         ptstat(actuel) = 0.0
         stol = sto
         call verist (stol)
         go to 999
  400 conten(sto) = 0
      call eerror(406)
c
  999 continue
      return
      end
```

```
c=============================================================
c
c
c
c     FACILITIES
c
c
c
c=============================================================
      integer function newfac (nomfac)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),occup(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq
      real tpus,tpini,lstpus
      real time,evtime(1)
      real ptstat
      dimension pt(1),lsc(1),prior(1),ptstat(1)
      equivalence (list,pt),(occup,lsc),(bidon,prior)
      equivalence (ptname,ptstat)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
      dimension nomfac(3)
c
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- new facility')
      if (old(facily) .eq. none) go to 100
         i=old(facily)
         old(facily)=suc(i)
         go to 300
c
  100 continue
      if (objlib + size(facily) .le. maxobj) go to 200
         call eerror(501)
         newfac = none
         go to 999
c
  200 continue
      i=objlib
      objlib=objlib+size(facily)
c
  300 continue
      newfac=i
      occup(i)=none
      class(i)=facily
      suc(i)=i
      pred(i)=i
```

```
      list(i)=lastfc
      lastfc=i
      do 350 j=1,3
         ptname(i,j) = nomfac(j)
  350 continue
      nbent(i) = 0
      nulent(i) = 0
      minwt(i) = 100000.0
      maxwt(i) = 0.0
      avwt(i) = 0.0
      stdwt(i) = 0.0
      nbwt(i) = 0
      inicpq(i) = 0
      mincpq(i) = "777777777
      maxcpq(i) = 0
      avcpq(i) = 0.0
      stdcpq(i) = 0.0
      nbcpq(i) = 0.0
      tpscpq(i) = time
      curcpq(i) = 0
      minst(i) = 100000.0
      maxst(i) = 0.0
      avst(i) = 0.0
      stdst(i) = 0.0
      nbst(i) = 0
      tpus(i) = 0.0
      tpini(i) = time
      lstpus(i) = time
c
  999 continue
      return
      end
c*************************************************************
      subroutine entfac (fac)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),occup(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq
      real tpus,tpini,lstpus,time
      real ptstat
      dimension pt(1),lsc(1),prior(1),ptstat(1)
      equivalence (list,pt),(occup,lsc),(bidon,prior)
      equivalence (ptname,ptstat)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /lst/ laststst,lastfc,lastgr,lastrg,lasths
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- entfac')
```

```
      if (class(fac) .eq. facily) go to 200
         call eerror (502)
         go to 999
  200 continue
      if (occup(fac) .ne. actuel) go to 400
         call eerror(503)
         go to 999
  400 continue
      nbent(fac) = nbent(fac) + 1
      ptstat(actuel) = time
      if (occup(fac) .ne. none) go to 600
         nulent(fac) = nulent(fac) + 1
         occup(fac) = actuel
         lstpus(fac) = time
         go to 999
  600 continue
      call printo(actuel,fac)
      call sisave(lsc(actuel))
      call return (pt(actuel))
      pt(actuel) = none
      call resume
c
  999 continue
      return
      end
c*****************************************************************
      subroutine leafac (fac)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),occup(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq
      real tpus,tpini,lstpus,time
      real ptstat
      dimension pt(1),lsc(1),prior(1),ptstat(1)
      equivalence (list,pt),(occup,lsc),(bidon,prior)
      equivalence (ptname,ptstat)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
c
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- leafac')
      if (class(fac) .eq. facily ) go to 200
         call eerror(504)
         go to 999
  200 continue
      if (occup(fac) .eq. actuel) go to 400
         call eerror(505)
         go to 999
```

```
  400 continue
      call addflt(avst(fac),stdst(fac),nbst(fac),minst(fac),
     1            maxst(fac),time-ptstat(actuel))
      ptstat(actuel) = 0.0
      occup(fac) = first(fac)
      if (occup(fac) .eq. none) go to 600
         call addcap(avcpq(fac),stdcpq(fac),nbcpq(fac),
     1               mincpq(fac),maxcpq(fac),curcpq(fac),
     2               time-tpscpq(fac))
         tpscpq(fac) = time
         call out(first(fac))
         call addflt(avwt(fac),stdwt(fac),nbwt(fac),
     1               minwt(fac),maxwt(fac),time-ptstat(occup(fac)))
         ptstat(occup(fac)) = time
         curcpq(fac) = curcpq(fac) - 1
         call activ (occup(fac),delay,0.0)
  600 continue
      tpus(fac) = tpus(fac) + (time-lstpus(fac))
      lstpus(fac) = time
c
  999 continue
      return
      end
```

```
c=====================================================================
c
c
c
c     GROUPES
c
c
c
c=====================================================================
      integer function newgr (capac)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      dimension pt(1),lsc(1),prior(1)
      equivalence (list,pt),(cap,lsc),(conten,prior)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststl,lastfc,lastgr,lastrg,lasths
c
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- new groupe')
      if (old(groupe) .eq. none) go to 100
         i=old(groupe)
         old(groupe)=suc(i)
         go to 300
c
  100 continue
      if (objlib + size(groupe) .le. maxobj) go to 200
         call eerror(601)
         newgr = none
         go to 999
c
  200 continue
      i=objlib
      objlib=objlib+size(groupe)
c
  300 continue
      newgr=i
      class(i)=groupe
      suc(i)=i
      pred(i)=i
      list(i)=lastgr
      lastgr=i
      cap(i)=capac
      conten(i)=0
c
  999 continue
      return
      end
c**********************************************************
      subroutine join (gr)
c
      implicit integer (a-z)
```

```
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1)
      common /objatr/ conten(1),ptstat(1)
      real time,evtime(1)
      dimension pt(1),lsc(1),prior(1)
      equivalence (list,pt),(cap,lsc),(conten,prior)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
      common /temgr/ p,pl
c
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- join')
      if (class(gr) .eq. groupe) go to 200
         call eerror(602)
         go to 999
  200 continue
      conten (gr) = conten (gr) + 1
      if ( conten(gr) .ge. cap(gr) ) go to 300
c  code en ligne de call wait (gr)
         call sisave (lsc(actuel))
         call return (pt(actuel))
         call into (actuel,gr)
         call resume
         go to 999
  300 continue
      conten (gr) = 0
      p = first (gr)
  350 continue
      if ( p .eq. gr ) go to 400
         pl = p
         call activ (p,delay,0.001)
         if (itrace .gt. 0) write (output,352)
  352    format (1x,'-- sortie join')
         p = suc (p)
         call out (pl)
         go to 350
  400 continue
      call hold (0.0)
c
  999 continue
      return
      end
```

```
c=============================================================
c
c
c
c       REGIONS
c
c
c
c=============================================================
      integer function newreg (nomreg)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),curct(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nbout(1),tpini(1)
      common /objatr/ mintt(1),maxtt(1),avtt(1),stdtt(1),nbtt(1)
      common /objatr/ inict(1),minct(1),maxct(1),avct(1),stdct(1)
      common /objatr/ tpsct(1),nbtct(1)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
      real time,ptstat,tpini,tpsct,nbtct
      real mintt,maxtt,avtt,stdtt,avct,stdct
      dimension ptstat(1)
      dimension nomreg(3)
      equivalence (ptname,ptstat)
c
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- new region')
      if (old(region) .eq. none) go to 100
           i = old(region)
           old(region) = suc(i)
           go to 300
c
  100 continue
      if (objlib + size(region) .le. maxobj) go to 200
           call eerror(701)
           newreg = none
           go to 999
c
  200 continue
      i = objlib
      objlib = objlib + size(region)
c
  300 continue
      newreg = i
      class(i) = region
      pred(i) = none
      suc(i) = none
      list(i) = lastrg
      lastrg = i
      do 350 j=1,3
           ptname(i,j) = nomreg(j)
           nomreg(j) = 0
  350 continue
      curct(i) = 0
```

```
      nbent(i) = 0
      nbout(i) = 0
      tpini(i) = time
      mintt(i) = 100000.0
      maxtt(i) = 0.0
      avtt(i) = 0.0
      stdtt(i) = 0.0
      nbtt(i) = 0
      inict(i) = 0
      minct(i) = "777777777
      maxct(i) = 0
      avct(i) = 0.0
      stdct(i) = 0.0
      tpsct(i) = time
      nbtct(i) = 0.0
c
  999 continue
      return
      end
c*****************************************************************
      subroutine entreg (reg)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),curct(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nbout(1),tpini(1)
      common /objatr/ mintt(1),maxtt(1),avtt(1),stdtt(1),nbtt(1)
      common /objatr/ inict(1),minct(1),maxct(1),avct(1),stdct(1)
      common /objatr/ tpsct(1),nbtct(1)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststs,lastfc,lastgr,lastrg,lasths
      real time,ptstat,tpini,tpsct,nbtct
      real mintt,maxtt,avtt,stdtt,avct,stdct
      dimension ptstat(1)
      dimension nomreg(3)
      equivalence (ptname,ptstat)
c
      if (itrace .gt. 0) write (output,5)
    5 format (1x,'-- entreg')
      if (class(reg) .eq. region) go to 200
           call eerror(702)
           go to 999
  200 continue
      nbent(reg) = nbent(reg) + 1
      ptstat(actuel) = time
      call addcap(avct(reg),stdct(reg),nbtct(reg),
     1            minct(reg),maxct(reg),curct(reg),
     2            time-tpsct(reg))
      tpsct(reg) = time
      curct(reg) = curct(reg) + 1
c
  999 continue
      return
      end
c*****************************************************************
```

```
      subroutine leareg (reg)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),curct(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nbout(1),tpini(1)
      common /objatr/ mintt(1),maxtt(1),avtt(1),stdtt(1),nbtt(1)
      common /objatr/ inict(1),minct(1),maxct(1),avct(1),stdct(1)
      common /objatr/ tpsct(1),nbtct(1)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lastbs
      real time,ptstat,tpini,tpsct,nbtct
      real mintt,maxtt,avtt,stdtt,avct,stdct
      dimension ptstat(1)
      dimension nomreg(3)
      equivalence (ptname,ptstat)
c
      if (itrace .gt. 0) write (output,5)
    5 format(1x,'-- leareg')
      if (class(reg) .eq. region) goto 200
           call eerror(704)
           go to 999
  200 continue
      nbout(reg) = nbout(reg) + 1
      call addflt(avtt(reg),stdtt(reg),nbtt(reg),mintt(reg),
     1            maxtt(reg),time-ptstat(actuel))
      call addcap(avct(reg),stdct(reg),nbtct(reg),
     1            minct(reg),maxct(reg),curct(reg),
     2            time-tpsct(reg))
      curct(reg) = curct(reg) - 1
      tpsct(reg) = time
c
  999 continue
      return
      end
```

```
c=====================================================================
c
c
c
c    HISTOGRAMMES
c
c
c
c=====================================================================
      integer function newhis (nomhis,nbint,borinf,taille)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /objatr/ class(1),pred(1),suc(1),list(1)
      common /objatr/ hnb(1),hbinf(1),hlong(1),hentr(1)
      common /objatr/ hmin(1),hmax(1),hsum(1),hvect(1,132)
      common /objatr/ ptname(1,3)
      common /io/ input,output
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststs,lastfc,lastgr,lastrg,lasths
      real borinf,taille
      real hbinf,hlong
      real time
      real hmin,hmax,hsum
      dimension nomhis(3)
c
      if (itrace .gt. 0) write (output,10)
   10 format (1x,'-- new histo')
      if ( old(histo) .eq. none ) go to 100
         i = old(histo)
         old(histo) = suc(i)
         go to 300
  100 continue
      if (objlib + size(histo) .le. maxobj ) go to 200
         call eerror (801)
         newhis = none
         go to 999
  200 continue
      i=objlib
      objlib=objlib + size(histo)
  300 continue
      newhis = i
      class(i) = histo
      suc(i) = i
      pred(i) = i
      list(i) = lasths
      lasths = i
      hnb(i) = nbint
      hbinf(i) = borinf
      hlong(i) = taille
      hmin(i) = 0.0
      hmax(i) = 0.0
      hsum(i) = 0.0
c
      do 400 boucle = 1,132,1
            hvect (i,boucle) = 0
```

```
  400        continue
c
      do 500 j = 1,3
             ptname (i,j) = nomhis (j)
  500        continue
c
  999 continue
      return
      end
c*********************************************************************
      subroutine addhis (hist,val)
c
      implicit integer (a-z)
      common /objatr/ class(1),pred(1),suc(1),list(1)
      common /objatr/ hnb(1),hbinf(1),hlong(1),hentr(1)
      common /objatr/ hmin(1),hmax(1),hsum(1),hvect(1,132)
      common /gpsss/  storag,facily,groupe,region,histo
      real val
      real hmin,hmax,hsum
      real hbinf,hlong
c
      if (class(hist) .eq. histo) go to 90
         call eerror (802)
         go to 999
   90 continue
      hsum(hist) = hsum(hist) + val
      hentr(hist) = hentr(hist) + 1
      if ( hentr(hist) .eq. 1 ) go to 100
         if ( val .lt. hmin(hist) ) go to 50
            if ( val .gt. hmax(hist) ) hmax(hist) = val
               go to 200
   50    continue
         hmin(hist) = val
         go to 200
  100 continue
      hmin(hist) = val
      hmax(hist) = val
  200 continue
      index = int((val-hbinf(hist))/hlong(hist))
      if ( index .lt. 0 ) go to 400
         if ( index .gt. hnb(hist) ) go to 350
            hvect(hist,index+2) = hvect(hist,index+2) + 1
            go to 999
  350    continue
         hvect(hist,hnb(hist)+2) = hvect(hist,hnb(hist)+2) + 1
         go to 999
  400 continue
      hvect(hist,1) = hvect(hist,1) + 1
c
  999 continue
      return
      end
c*********************************************************************
      subroutine prthis (hist)
c
      implicit integer (a-z)
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1)
      common /objatr/ hnb(1),hbinf(1),hlong(1),hentr(1)
```

```
      common /objatr/ hmin(1),hmax(1),hsum(1),hvect(1,132)
      common /objatr/ ptname(1,3)
      common /gpsss/  storag,facily,groupe,region,histo
      real hmin,hmax,hsum
      real hbinf,hlong
      real a,b,c
      dimension name(3)
c
      if (class(hist) .eq. histo) go to 90
         call eerror (803)
         go to 999
   90 continue
c
      do 92 j = 1,3
            name (j) = ptname (hist,j)
    92      continue
         write (output,5),name
         write (output,6)
c
    95 continue
      boucle = 1
      bclsup = hnb(hist) + 1
c
c     a = -infini
      b = hbinf(hist)
      c = hlong(hist)
c
      write (output,10),boucle,b,hvect(hist,boucle)
c
      do 100 boucle=2,bclsup,1
            a = b
            b = b + c
            write (output,20),boucle,a,b,hvect(hist,boucle)
   100      continue
c
      boucle = bclsup + 1
      a = b
c     b = + infini
      write (output,30),boucle,a,hvect(hist,boucle)
c
     5 format (1h1,10x,'Histogramme :   ',3a5)
     6 format (1x,8x,'**********************************')
c
    10 format (1h-,' Intervalle num : ',i3
     1        ,'      ] - infini , ',f8.3,' ] '
     2        ,' nombre d''occurences : ',i5)
    20 format (1x,' Intervalle num : ',i3
     1        ,'      ] ',f8.3,' , ',f8.3,' ] '
     2        ,' nombre d''occurences : ',i5)
    30 format (1x,' Intervalle num : ',i3
     1        ,'      ] ',f8.3,' , + infini [ '
     2        ,' nombre d''occurences : ',i5)
c
   999 continue
      return
      end
c*************************************************************
      subroutine hisrep
c
```

```
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /lst/ laststst,lastfc,lastgr,lastrg,lasths
      common /objatr/ class(1),pred(1),suc(1),list(1)
      common /objatr/ hnb(1),hbinf(1),hlong(1),hentr(1)
      common /objatr/ hmin(1),hmax(1),hsum(1),hvect(1,132)
      common /objatr/ ptname(1,3)
c
      write (output,101)
      write (output,102)
      write (output,103)
      write (output,104)
      write (output,105)
c
      if (lasths .ne. none) go to 500
          write (output,50)
          write (output,51)
          go to 999
  500 continue
      point = lasths
  600 continue
      if (point .eq. none) go to 999
         call prthis (point)
         point = list(point)
         go to 600
c
   50 format (1h-,10x,'Ce programme n''a pas genere d''histogrammes !')
   51 format (1x, 10x,'*********************************************')
  101 format (1h1,10x,'*********************************************')
  102 format (1x ,10x,'*                                           *')
  103 format (1x ,10x,'*     D:   Histogrammes                     *')
  104 format (1x ,10x,'*                                           *')
  105 format (1x ,10x,'*********************************************')
c
  999 continue
      return
      end
```

```
c=====================================================================
c
c
c
c     NOMBRES ALEATOIRES
c
c
c
c=====================================================================
      real function rand(u)
c
      integer u,modulo
      data modulo /67099547/
c
      u = mod(u*(2**5),modulo)
      u = mod(u*(2**5),modulo)
      u = mod(u*(2**3),modulo)
      rand = float(u)/modulo
c
  999 continue
      return
      end
c**********************************************************************
      logical function draw (a,u)
c
      integer u
      real rand,a,b
c
      draw = .false.
      b = rand(u)
      if (b .lt. a) draw = .true.
c
  999 continue
      return
      end
c**********************************************************************
      real function unif (a,b,u)
c
      integer u
      real rand,a,b
c
      if (b .gt. a) go to 200
         call eerror(901)
         go to 999
  200 continue
      unif = a + (b-a)*rand(u)
c
  999 continue
      return
      end
c**********************************************************************
      integer function randin(a,b,u)
c
      integer a,b,u
      real unif,result,al,bl
c
      if (b .ge. a) go to 200
         call eerror(902)
         go to 999
```

```
  200 continue
      al = float(a)
      bl = float(b) + 1.0
      result = unif(al,bl,u)
      randin = ifix(result)
c
  999 continue
      return
      end
c*****************************************************************
      real function negexp (a,u)
c
      real a,rand
      integer u
c
      negexp = -alog(rand(u))/a
c
  999 continue
      return
      end
c*****************************************************************
      real function normal (a,b,u)
c
      real a,b
      integer u
      real somme,r
      integer i
c
      somme = 0.0
      do 100 i=1,12,1
             r=rand(u)
             somme = somme + r
  100        continue
      normal = b * ( somme - 6.0 ) + a
c
  999 continue
      return
      end
c*****************************************************************
      integer function poissn (a,u)
c
      implicit integer (a-z)
      real a,b,tr,r,rand
      poissn = 0
      b = exp(-a)
      tr = 1.0
   50 continue
      r = rand(u)
      tr = tr * r
      if ( tr .lt. b ) go to 999
         poissn = poissn + 1
         go to 50
c
  999 continue
      return
      end
c*****************************************************************
      real function gamma (k,a,u)
c
```

```
      integer k,u
      real a,tr,r
      tr = 1.0
      do 50 i=1,k
            r=rand(u)
            tr = tr * r
   50       continue
      gamma = -alog (tr) / a
c
  999 continue
      return
      end
c*********************************************************************
      integer function hypgeo (nbpop,nbech,p,u)
c
      implicit integer (a-z)
      real p,r,rand,ptemp,s
      n1 = nbpop
      n2 = nbech
      hypgeo = 0
      do 100 i=1,n2,1
            r = rand (u)
            if ( r .gt. ptemp ) go to 40
               s = 1.0
               hypgeo = hypgeo + 1
               go to 60
   40       continue
            s = 0.0
   60       continue
            ptemp = ( n1 * p - s ) / ( n1 - 1 )
            n1 = n1 - 1
  100       continue
c
  999 continue
      return
      end
c*********************************************************************
      integer function binom (n,p,u)
c
      implicit integer (a-z)
      real r,rand,p
      binom = 0
      do 100 i = 1,n
            r = rand (u)
            if ( r .ge. p ) go to 100
               binom = binom + 1
  100       continue
c
  999 continue
      return
      end
```

```
c=====================================================================
c
c
c
c   DEBUGGING, TRACAGE ET ERREURS
c
c
c
c=====================================================================
      subroutine werror (n)
c
      implicit integer (a-z)
      common /io/ input,output
      common /err/ nerr
c
      nerr=nerr+1
      if (nerr .le. 5) go to 50
         write (output,98)
         write (output,99)
         stop
   50 continue
      write (output,100),n
  100 format (//////////,1x,'*** warning : ',i5)
c
c
      if (n .eq. 101) go to 101
      if (n .eq. 201) go to 201
      if (n .eq. 202) go to 202
      if (n .eq. 203) go to 203
      if (n .eq. 204) go to 204
      if (n .eq. 205) go to 205
      if (n .eq. 206) go to 206
      if (n .eq. 207) go to 207
      if (n .eq. 208) go to 208
      if (n .eq. 209) go to 209
      go to 998
c
c
c
c
  101 write (output,1101)
      write (output,2101)
      write (output,3101)
      write (output,4101)
      write (output,5101)
      go to 999
c
  201 write (output,1201)
      write (output,2201)
      go to 999
c
  202 write (output,1202)
      write (output,2202)
      write (output,3202)
      go to 999
c
  203 write (output,1203)
      write (output,2203)
      write (output,3203)
```

```
      go to 999
c
  204 write (output,1204)
      write (output,2204)
      write (output,3204)
      write (output,4204)
      go to 999
c
  205 write (output,1205)
      write (output,2205)
      go to 999
c
  206 write (output,1206)
      write (output,2206)
      write (output,3206)
      write (output,4206)
      go to 999
c
  207 write (output,1207)
      write (output,2207)
      write (output,3207)
      go to 999
c
  208 write (output,1208)
      write (output,2208)
      write (output,3208)
      write (output,4208)
      go to 999
c
  209 write (output,1209)
      write (output,2209)
      write (output,3209)
      go to 999
c
  998 write (output,1998)
      write (output,2998)
      write (output,3998)
      go to 999
c
   98 format( 51h Plus de 5 avertissements ont ete imprimes !       )
   99 format( 51h ....Arret premature de la simulation...           )
 1101 format( 51h cause : essai de detruire a l'aide de Kill        )
 2101 format( 51h        une tete de liste, un storage, une facilite)
 3101 format( 51h        ou un groupe alors qu'il reste encore des  )
 4101 format( 51h        des objets dans la liste.                  )
 5101 format( 51h resultat : aucun.                                 )
 1201 format( 51h cause : activation d'un objet none                )
 2201 format( 51h resultat : aucun.                                 )
 1202 format( 51h cause : activation d'un objet avec un delai       )
 2202 format( 51h            inferieur a 0                          )
 3202 format( 51h resultat : activation de l'objet avec un delai    )
 4202 format( 51h                      nul                         )
 1203 format( 51h cause : activation d'un objet a un temps inferieur)
 2203 format( 51h                      au temps courant            )
 3203 format( 51h resultat : activation de l'objet au temps courant )
 1204 format( 51h cause : activation d'un objet avant ou apres un   )
 2204 format( 51h                      objet qui ne se trouve pas  )
 3204 format( 51h                      dans la sqs                 )
 4204 format( 51h resultat : aucun.                                 )
```

```
 1205 format( 51h cause : activation d'un objet none                )
 2205 format( 51h resultat : aucun.                                 )
 1206 format( 51h cause : activation d'un objet avec un delai       )
 2206 format( 51h            inferieur a 0                          )
 3206 format( 51h resultat : activation de l'objet avec un delai    )
 4206 format( 51h                    nul                            )
 1207 format( 51h cause : activation d'un objet a un temps inferieur)
 2207 format( 51h                    au temps courant               )
 3207 format( 51h resultat : activation de l'objet au temps courant )
 1208 format( 51h cause : appel a la routine Hold avec un delai     )
 2208 format( 51h            inferieur a 0.0                        )
 3208 format( 51h resultat : appel a la routine Hold avec un delai  )
 4208 format( 51h            egal a 0.0                             )
 1209 format( 51h cause : destruction d'un objet qui ne se trouve   )
 2209 format( 51h            pas dans la SOS                        )
 3209 format( 51h resultat : aucun                                  )
 1998 format( 51h cause : appel a la routine WERROR avec comme      )
 2998 format( 51h            argument une valeur inconnue.          )
 3998 format( 51h resultat : aucun                                  )
c
c
  999 continue
      return
      end
c*****************************************************************
      subroutine eerror (n)
c
      implicit integer (a-z)
      common /io/ input,output
c
      write (output,10),n
   10 format (/////,1x,'*** error : ',i5,////)
c
c
      if (n .eq. 101) go to 101
      if (n .eq. 102) go to 102
      if (n .eq. 103) go to 103
      if (n .eq. 104) go to 104
      if (n .eq. 105) go to 105
      if (n .eq. 106) go to 106
      if (n .eq. 107) go to 107
      if (n .eq. 110) go to 110
      if (n .eq. 201) go to 201
      if (n .eq. 202) go to 202
      if (n .eq. 203) go to 203
      if (n .eq. 204) go to 204
      if (n .eq. 205) go to 205
      if (n .eq. 301) go to 301
      if (n .eq. 302) go to 302
      if (n .eq. 303) go to 303
      if (n .eq. 304) go to 304
      if (n .eq. 310) go to 310
      if (n .eq. 401) go to 401
      if (n .eq. 402) go to 402
      if (n .eq. 403) go to 403
      if (n .eq. 404) go to 404
      if (n .eq. 405) go to 405
      if (n .eq. 406) go to 406
      if (n .eq. 501) go to 501
```

```
      if (n .eq. 502) go to 502
      if (n .eq. 503) go to 503
      if (n .eq. 504) go to 504
      if (n .eq. 505) go to 505
      if (n .eq. 601) go to 601
      if (n .eq. 602) go to 602
      if (n .eq. 701) go to 701
      if (n .eq. 702) go to 702
      if (n .eq. 704) go to 704
      if (n .eq. 801) go to 801
      if (n .eq. 802) go to 802
      if (n .eq. 803) go to 803
      if (n .eq. 901) go to 901
      if (n .eq. 902) go to 902
      if (n .eq. 950) go to 950
      if (n .eq. 951) go to 951
      if (n .eq. 952) go to 952
      go to 998
c
c
  101 write (output,1101)
      write (output,2101)
      write (output,3101)
      go to 999
c
  102 write (output,1102)
      write (output,2102)
      write (output,3102)
      go to 999
c
  103 write (output,1103)
      write (output,2103)
      write (output,3103)
      write (output,4103)
      write (output,5103)
      write (output,6103)
      write (output,7103)
      write (output,8103)
      write (output,9103)
      go to 999
c
  104 write (output,1104)
      write (output,2104)
      write (output,3104)
      go to 999
c
  105 write (output,1105)
      write (output,2105)
      write (output,3105)
      go to 999
c
  106 write (output,1106)
      write (output,2106)
      write (output,3106)
      go to 999
c
  107 write (output,1107)
      write (output,2107)
      write (output,3107)
```

```
      write (output,4107)
      go to 999
c
  110 write (output,1110)
      write (output,2110)
      write (output,3110)
      write (output,4110)
      go to 999
c
  201 write (output,1201)
      write (output,2201)
      write (output,3201)
      write (output,4201)
      go to 999
c
  202 write (output,1202)
      write (output,2202)
      write (output,3202)
      write (output,4202)
      write (output,5202)
      go to 999
c
  203 write (output,1203)
      write (output,2203)
      write (output,3203)
      write (output,4203)
      go to 999
c
  204 write (output,1204)
      write (output,2204)
      write (output,3204)
      go to 999
c
  205 write (output,1205)
      write (output,2305)
      write (output,3205)
      write (output,4205)
      write (output,5205)
      write (output,6205)
      go to 999
c
  301 write (output,1301)
      write (output,2301)
      write (output,3301)
      go to 999
c
  302 write (output,1302)
      write (output,2302)
      write (output,3302)
      write (output,4302)
      write (output,5302)
      write (output,6302)
      write (output,7302)
      write (output,8302)
      go to 999
c
  303 write (output,1303)
      write (output,2303)
      write (output,3303)
```

```
      write (output,4303)
      go to 999
c
  304 write (output,1304)
      write (output,2304)
      write (output,3304)
      write (output,4304)
      go to 999
c
  310 write (output,1310)
      write (output,2310)
      write (output,3310)
      write (output,4310)
      go to 999
c
  401 write (output,1401)
      write (output,2401)
      write (output,3401)
      write (output,4401)
      go to 999
c
  402 write (output,1402)
      write (output,2402)
      write (output,3402)
      go to 999
c
  403 write (output,1403)
      write (output,2403)
      write (output,3403)
      write (output,4403)
      go to 999
c
  404 write (output,1404)
      write (output,2404)
      write (output,3404)
      go to 999
c
  405 write (output,1405)
      write (output,2405)
      write (output,3405)
      write (output,4405)
      go to 999
c
  406 write (output,1406)
      write (output,2406)
      write (output,3406)
      write (output,4406)
      write (output,5406)
      write (output,6406)
      go to 999
c
  501 write (output,1501)
      write (output,2501)
      write (output,3501)
      write (output,4501)
      go to 999
c
  502 write (output,1502)
      write (output,2502)
```

```
      write (output,3502)
      go to 999
c
  503 write (output,1503)
      write (output,2503)
      write (output,3503)
      go to 999
c
  504 write (output,1504)
      write (output,2504)
      write (output,3504)
      go to 999
c
  505 write (output,1505)
      write (output,2505)
      write (output,3505)
      write (output,4505)
      write (output,5505)
      go to 999
c
  601 write (output,1601)
      write (output,2601)
      write (output,3601)
      write (output,4601)
      go to 999
c
  602 write (output,1602)
      write (output,2602)
      write (output,3602)
      go to 999
c
  701 write (output,1701)
      write (output,2701)
      write (output,3701)
      write (output,4701)
      go to 999
c
  702 write (output,1702)
      write (output,2702)
      write (output,3702)
      go to 999
c
  704 write (output,1704)
      write (output,2704)
      write (output,3704)
      go to 999
c
  801 write (output,1801)
      write (output,2801)
      write (output,3801)
      write (output,4801)
      go to 999
c
  802 write (output,1802)
      write (output,2802)
      write (output,3802)
      go to 999
c
  803 write (output,1803)
```

```
      write (output,2803)
      write (output,3803)
      go to 999
c
  901 write (output,1901)
      write (output,2901)
      write (output,3901)
      go to 999
c
  902 write (output,1902)
      write (output,2902)
      write (output,3902)
      go to 999
c
  950 write (output,1950)
      write (output,2950)
      write (output,3950)
      go to 999
c
  951 write (output,1951)
      write (output,2951)
      write (output,3951)
      go to 999
c
  952 write (output,1952)
      write (output,2952)
      write (output,3952)
      go to 999
c
  998 write (output,1998)
      write (output,2998)
      write (output,3998)
      go to 999
c
c
c
 1101 format( 51h cause : nombre de classes superieur au nombre     )
 2101 format( 51h            maximal autorise.                      )
 3101 format( 51h resultat : arret de la simulation.                )
 1102 format( 51h cause : appel a la fonction Class avec l'argument )
 2102 format( 51h            taille invalide.                       )
 3102 format( 51h resultat : arret de la simulation.                )
 1103 format( 51h cause : manque de place dans la zone d'adressage  )
 2103 format( 51h            dynamique : tout l'espace disponible   )
 3103 format( 51h            a deja ete alloue.                     )
 4103 format( 51h resultat : arret de la simulation.                )
 5103 format( 51h remede possible : verifiez si vous avez bien      )
 6103 format( 51h            detruit tous vos objets qui ne seront  )
 7103 format( 51h            plus references dans le futur . N'avez )
 8103 format( 51h            -vous pas oublie d'appeler la routine  )
 9103 format( 51h            Endpro a la fin de vos routines        )
 1104 format( 51h cause : essai de destruction de l'objet NONE.     )
 2104 format( 51h          (cet objet ne peut jamais etre detruit !) )
 3104 format( 51h resultat : arret de la simulation                 )
 1105 format( 51h cause : appel a la fonction Idle avec  un argument)
 2105 format( 51h          different d'un processus.                )
 3105 format( 51h resultat : arret de la simulation.                )
 1106 format( 51h cause : appel a la fonction Evtime avec un        )
 2106 format( 51h          argument different d'un processus.       )
```

```
3106 format( 51h resultat : arret de la simulation.                )
1107 format( 51h cause : appel errone a la fonction Evtime : le    )
2107 format( 51h         processus dont vous donnez l'indice n'est )
3107 format( 51h         pas un processus actif ou suspendu.       )
4107 format( 51h resultat : arret de la simulation.                )
1110 format( 51h cause : appel errone a la sous routine Kill       )
2110 format( 51h         obj ne peut pas referencer le processus   )
3110 format( 51h         decrivant le programme principal.         )
4110 format( 51h resulat : arret de la simulation.                 )
1201 format( 51h cause : activation d'un objet qui se trouve deja  )
2201 format( 51h            dans la SQS.                           )
3201 format( 51h resulat : arret de la simulation.                 )
4201 format( 51h remede : utiliser la sous routine React.          )
1202 format( 51h cause : appel a la sous routine Activ avec un     )
2202 format( 51h            code non admis.                        )
3202 format( 51h            ( les codes valables sont : Delay,     )
4202 format( 51h              At, Before, After.  )                )
5202 format( 51h resultat : arret de la simulation.                )
1203 format( 51h cause : appel a la sous routine Pactiv avec un    )
2203 format( 51h            code non admis ( les codes valables    )
3203 format( 51h            sont : Delay, At.  )                   )
4203 format( 51h resultat : arret de la simulation.                )
1204 format( 51h cause : activation d'un objet qui se trouve deja  )
2204 format( 51h            dans la SQS.                           )
3204 format( 51h resultat : arret de la simulation.                )
1205 format( 51h cause : erreur systeme.                           )
2305 format( 51h            ( Cette erreur ne devrait pas se       )
3205 format( 51h              produire si vous ne modifiez         )
4205 format( 51h              pas les indices que le systeme vous  )
5205 format( 51h              renvoie.  )                          )
6205 format( 51h resultat : arret de la simulation.                )
1301 format( 51h cause : tentative de retire une tete de liste de  )
2301 format( 51h            la liste a laquelle elle appartient.   )
3301 format( 51h resultat : arret de la simulation                 )
1302 format( 51h cause : tentative erronee d'inserer un objet Obj1 )
2302 format( 51h            avant ou apres un autre objet Obj2     )
3302 format( 51h          erreurs possibles :                      )
4302 format( 51h           - Obj2 n'est pas une tete de liste      )
5302 format( 51h           - Obj2 n'appartient pas a une liste     )
6302 format( 51h           - Obj1 est une tete de liste ou l'objet )
7302 format( 51h             NONE                                  )
8302 format( 51h resultat: arret de la simulation.                 )
1303 format( 51h cause : tentative d'inserer un objet dans autre   )
2303 format( 51h            chose qu'une liste, un storage, une    )
3303 format( 51h            facility ou un groupe.                 )
4303 format( 51h resultat : arret de la simulation.                )
1304 format( 51h cause : appel a la fonction Empty avec un         )
2304 format( 51h            argument different d'une liste, un     )
3304 format( 51h            storage, une facility, un groupe.      )
4304 format( 51h resultat : arret de la simulation.                )
1310 format( 51h cause : appel a la fonction Cardi avec un         )
2310 format( 51h            argument different d'une liste, un     )
3310 format( 51h            storage, une facility, un groupe.      )
4310 format( 51h resultat : arret de la simulation.                )
1401 format( 51h cause : manque de place dans la zone d'adressage  )
2401 format( 51h            dynamique : tout l'espace disponible   )
3401 format( 51h            a deja ete alloue.                     )
4401 format( 51h resultat : arret de la simulation.                )
1402 format( 51h cause : appel a la routine Entst avec le premier  )
```

```
2402 format( 51h            argument different d'un storage.        )
3402 format( 51h resultat : arret de la simulation.                 )
1403 format( 51h cause : appel a la routine Entst avec le deuxieme )
2403 format( 51h            argument superieur a la capacite        )
3403 format( 51h            maximale du storage.                    )
4403 format( 51h resultat : arret de la simulation.                 )
1404 format( 51h cause : appel a la routine Least avec le premier  )
2404 format( 51h            argument different d'un storage         )
3404 format( 51h resultat : arret de la simulation.                 )
1405 format( 51h cause : appel a la routine Least avec le deuxieme )
2405 format( 51h            argument superieur a la capacite        )
3405 format( 51h            maximale du storage.                    )
4405 format( 51h resultat : arret de la simulation.                 )
1406 format( 51h cause : erreur systeme.                            )
2406 format( 51h            ( Cette erreur ne devrait pas se        )
3406 format( 51h              produire si vous ne modifiez          )
4406 format( 51h              pas les indices que le systeme vous   )
5406 format( 51h              renvoie.  )                           )
6406 format( 51h resultat : arret de la simulation.                 )
1501 format( 51h cause : manque de place dans la zone d'adressage  )
2501 format( 51h            dynamique : tout l'espace disponible    )
3501 format( 51h            a deja ete alloue.                      )
4501 format( 51h resultat : arret de la simulation.                 )
1502 format( 51h cause : appel a la routine Entfac avec un         )
2502 format( 51h            argument different d'une facility.      )
3502 format( 51h resultat : arret de la simulation.                 )
1503 format( 51h cause : deuxieme appel a la routine Entfac pour   )
2503 format( 51h            le meme objet et la meme facility.      )
3503 format( 51h resultat : arret de la simulation.                 )
1504 format( 51h cause : appel a la routine Leafac avec un         )
2504 format( 51h            argument different d'une facility.      )
3504 format( 51h resultat : arret de la simulation.                 )
1505 format( 51h cause : appel errone de la routine Leafac : le    )
2505 format( 51h            processus appelant n'est pas celui qui )
3505 format( 51h            se trouve presentement dans la          )
4505 format( 51h            facility.                               )
5505 format( 51h resultat : arret de la simulation.                 )
1601 format( 51h cause : manque de place dans la zone d'adressage  )
2601 format( 51h            dynamique : tout l'espace disponible    )
3601 format( 51h            a deja ete alloue.                      )
4601 format( 51h resultat : arret de la simulation.                 )
1602 format( 51h cause : appel a la routine Join avec un argument  )
2602 format( 51h            different d'un groupe.                  )
3602 format( 51h resultat : arret de la simulation.                 )
1701 format( 51h cause : manque de place dans la zone d'adressage  )
2701 format( 51h            dynamique : tout l'espace disponible    )
3701 format( 51h            a deja ete alloue.                      )
4701 format( 51h resultat : arret de la simulation.                 )
1702 format( 51h cause : appel a la routine Entreg avec un         )
2702 format( 51h            argument different d'une region.        )
3702 format( 51h resultat : arret de la simulation.                 )
1704 format( 51h cause : appel a la routine Leareg avec un         )
2704 format( 51h            argument different d'une region.        )
3704 format( 51h resultat : arret de la simulation.                 )
1801 format( 51h cause : manque de place dans la zone d'adressage  )
2801 format( 51h            dynamique : tout l'espace disponible    )
3801 format( 51h            a deja ete alloue.                      )
4801 format( 51h resultat : arret de la simulation.                 )
1802 format( 51h cause : appel a la routine Addhis avec le premier )
```

```
 2802 format( 51h            argument different d'un histogramme.   )
 3802 format( 51h resultat : arret de la simulation.                )
 1803 format( 51h cause : appel a la routine Prthis avec un argument)
 2803 format( 51h            different d'un histogramme.            )
 3803 format( 51h resultat : arret de la simulation.                )
 1901 format( 51h cause : appel errone a la fonction Unif(a,b,u)    )
 2901 format( 51h            condition obligatoire : b .gt. a .     )
 3901 format( 51h resultat : arret de la simulation.                )
 1902 format( 51h cause : appel errone a la fonction Randin(a,b,u)  )
 2902 format( 51h            condition obligatoire : b .gt. a .     )
 3902 format( 51h resultat : arret de la simulation.                )
 1950 format( 51h cause : appel a la routine Inireg avec un         )
 2950 format( 51h            argument different d'une region.       )
 3950 format( 51h resultat : arret de la simulation.                )
 1951 format( 51h cause : appel a la routine Inireg avec un         )
 2951 format( 51h            argument different d'une region.       )
 3951 format( 51h resultat : arret de la simulation.                )
 1952 format( 51h cause : appel a la routine Inireg avec un         )
 2952 format( 51h            argument different d'une region.       )
 3952 format( 51h resultat : arret de la simulation.                )
 1998 format( 51h cause : appel a la routine Error avec comme       )
 2998 format( 51h            argument une valeur inconnue.          )
 3998 format( 51h resultat : arret de simulation.                   )
c
c
  999 continue
      stop
      end
c*****************************************************************
      subroutine trace
c
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
        itrace = 1
        go to 999
c
      entry untrac
        itrace = 0
        go to 999
c
      entry tracex
        itrace = 2
c
  999 continue
      return
      end
c*****************************************************************
      subroutine suivre
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      real itime
      equivalence (evtime,lsc)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
```

```
      common /secret/ sqs,objlib,maxobj,notice,itrace
      logical empty
c
      write (output,98)
      write (output,99)
   98 format (/////,21h parcours de la sqs    )
   99 format (   21h ******************   ,//)
c
      if (empty(sqs)) go to 200
         i = 1
         point = suc(sqs)
  100    continue
         icur = pt(point)
         itime = evtime(point)
         iclass = class(icur)
         if (pt(pred(point)) .ne. none) go to 150
            ipred = pt(pred(pred(point)))
            go to 160
  150    continue
         ipred = pt(pred(point))
  160    continue
         if (pt(suc(point)) .ne. none) go to 170
            isuc = pt(suc(suc(point)))
            go to 180
  170    continue
         isuc = pt(suc(point))
  180    continue
         ilsc = lsc(icur)
c
         write (output,1),i
         write (output,2)
         write (output,3),icur
         write (output,4),itime
         write (output,5),iclass
         write (output,6),ipred
         write (output,7),isuc
         write (output,8),ilsc
c
    1    format (// 13h objet numero ,2x,i5)
    2    format (20h ******************   ,/)
    3    format (28h indice                          ,2x,o12)
    4    format (28h temps d'evenement :             ,2x,f8.3)
    5    format (28h numero de classe :              ,2x,i5)
    6    format (28h predecesseur dans la sqs :      ,2x,o12)
    7    format (28h successeur dans la sqs :        ,2x,o12)
    8    format (28h pointeur de reactivation :      ,2x,o12)
c
         i = i + 1
         point = suc (point)
         if (point .eq. sqs) go to 999
            go to 100
c
  200 continue
      write (output,201)
  201 format (//,17h la sqs est vide ,//)
c
  999 continue
      return
      end
```

```
c*************************************************************
      subroutine dumps
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),pt(1),lsc(1),prior(1)
      common /objatr/ ptstat(1)
      real time,evtime(1)
      equivalence (evtime,lsc)
      common /classv/ freecl,maxcl,size(20),lsc0(20),old(20)
      common /secret/ sqs,objlib,maxobj,notice,itrace
c
      write (output,10)
   10 format (//,1x,'** dump **',//)
      write (output,20),none
   20 format (1x,'none = ',o12)
      write (output,30),sqs
   30 format (1x,'sqs  = ',o12)
      write (output,40),actuel
   40 format (1x,'curr = ',o12)
      write (output,50),time
   50 format (1x,'time = ',f8.3)
      write (output,60)
   60 format (1x,' ')
      do 100 i=1,freecl
             write (output,70),i,size(i),lsc0(i),old(i)
   70        format (1x,i5,2x,i5,o12,2x,o12)
  100        continue
      write (output,80)
   80 format (1x,' ')
      i = 1
      izptr = none - 1 + i
  200 continue
      if (izptr .eq. objlib) go to 300
         write (output,90),i,izptr,class(izptr),class(izptr)
   90    format (1x,i5,2x,i5,2x,i11,2x,o12)
         i = i + 1
         izptr = izptr + 1
         go to 200
  300 continue
c
  999 continue
      return
      end
```

```
c=====================================================================
c
c
c
c   STATISTIQUES
c
c
c
c=====================================================================
      subroutine addflt (moy,var,compt,min,max,neuf)
c
      real moy,var,min,max,neuf
      integer compt
c
      moy = ((float(compt)*moy)+neuf)/float(compt+1)
      var = ((float(compt)*var)+(neuf*neuf))/float(compt+1)
      if (neuf .lt. min) min = neuf
      if (neuf .gt. max) max = neuf
      compt = compt + 1
c
  999 continue
      return
      end
c**********************************************************************
      subroutine addcap (cpmoy,cpvar,tptot,cpmin,cpmax,cpneuf,tpneuf)
c
      real cpmoy,cpvar,tptot,tpneuf
      integer cpmin,cpmax,cpneuf
c
      if (tpneuf .eq. 0.0) go to 999
      cpmoy = ((tptot*cpmoy)+(float(cpneuf)*tpneuf))/(tptot+tpneuf)
      cpvar = ((tptot*cpvar)+(float(cpneuf**2)*tpneuf))/(tptot+tpneuf)
      if (cpneuf .lt. cpmin) cpmin = cpneuf
      if (cpneuf .gt. cpmax) cpmax = cpneuf
      tptot = tptot + tpneuf
c
  999 continue
      return
      end
c**********************************************************************
      subroutine addint (moy,var,compt,min,max,neuf)
c
      real moy,var
      integer compt,min,max,neuf
c
      moy = ((float(compt)*moy)+float(neuf))/float(compt+1)
      var = ((float(compt)*var)+float(neuf**2))/float(compt+1)
      if (neuf .lt. min) min = neuf
      if (neuf .gt. max) max = neuf
      compt = compt + 1
c
  999 continue
      return
      end
c**********************************************************************
      subroutine storep
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
```

```
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      common /tmps/ time
      common /io/ input,output
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststst,lastfc,lastgr,lastrg,lasths
      dimension name(3)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
      real tpsobs,txus,ectyp,txnul
c
      write (output,101)
      write (output,102)
      write (output,103)
      write (output,104)
      write (output,105)
c
      if (laststst .ne. none) go to 2010
         write (output,151)
         write (output,152)
         go to 999
c
 2010 continue
      write (output,211)
      write (output,212)
c
      write (output,221)
      write (output,222)
      write (output,223)
      write (output,224)
      write (output,225)
      write (output,226)
      write (output,227)
c
      point = laststst
 2020 continue
      if (point .eq. none) go to 2030
         do 2100 i=1,3
            name(i) = ptname(point,i)
 2100    continue
         if (conten(point) .le. 0) go to 2025
            tpus(point) = tpus(point) + (time - lstpus(point))
            lstpus(point) = time
 2025    continue
         txnul = (float(nulent(point))/float(nbent(point))) * 100.0
         tpsobs = time - tpini(point)
         txus = (tpus(point) / tpsobs) * 100.0
         write (output,231),name,nbent(point),nulent(point),txnul,
     1                      tpsobs,tpus(point),txus
         point = list(point)
```

```
         go to 2020
c
 2030 continue
      write (output,228)
      write (output,229)
c
      write (output,311)
      write (output,312)
c
      write (output,321)
      write (output,322)
      write (output,323)
      write (output,324)
      write (output,325)
      write (output,326)
      write (output,327)
c
      point = laststt
 3020 continue
      if (point .eq. none) go to 3030
         do 3100 i=1,3
            name(i) = ptname(point,i)
 3100    continue
         ectyp = sqrt(stdrqs(point) - (avrqs(point)*avrqs(point)))
         write (output,331),name,nbrqs(point),minrqs(point),
     1                      maxrqs(point),avrqs(point),ectyp
         point = list(point)
         go to 3020
c
 3030 continue
      write (output,328)
      write (output,329)
c
      write (output,411)
      write (output,412)
c
      write (output,421)
      write (output,422)
      write (output,423)
      write (output,424)
      write (output,425)
      write (output,426)
      write (output,427)
c
      point = laststt
 4020 continue
      if (point .eq. none) go to 4030
         do 4100 i=1,3
            name(i) = ptname(point,i)
 4100    continue
         ectyp = sqrt(stdwt(point) - (avwt(point)*avwt(point)))
         write (output,431),name,nbwt(point),minwt(point),
     1                      maxwt(point),avwt(point),ectyp
         point = list(point)
         go to 4020
c
 4030 continue
      write (output,428)
      write (output,429)
```

```
c
      write (output,511)
      write (output,512)
c
      write (output,521)
      write (output,522)
      write (output,523)
      write (output,524)
      write (output,525)
      write (output,526)
      write (output,527)
c
      point = laststt
 5020 continue
      if (point .eq. none) go to 5030
         do 5100 i=1,3
            name(i) = ptname(point,i)
 5100    continue
         call addcap (avcpq(point),stdcpq(point),nbcpq(point),
     1                mincpq(point),maxcpq(point),curcpq(point),
     2                time-tpscpq(point))
         tpscpq(point) = time
         ectyp = sqrt(stdcpq(point) - (avcpq(point)*avcpq(point)))
         write (output,531),name,inicpq(point),mincpq(point),
     1                      maxcpq(point),avcpq(point),
     2                      ectyp,curcpq
         point = list(point)
         go to 5020
c
 5030 continue
      write (output,528)
      write (output,529)
c
      write (output,611)
      write (output,612)
c
      write (output,621)
      write (output,622)
      write (output,623)
      write (output,624)
      write (output,625)
      write (output,626)
      write (output,627)
c
      point = laststt
 6020 continue
      if (point .eq. none) go to 6030
         do 6100 i=1,3
            name(i) = ptname(point,i)
 6100    continue
         call addcap (avcps(point),stdcps(point),nbcps(point),
     1                mincps(point),maxcps(point),conten(point),
     2                time-tpscps(point))
         tpscps(point) = time
         ectyp = sqrt(stdcps(point) -(avcps(point)*avcps(point)))
         write (output,631),name,cap(point),inicps(point),
     1                      mincps(point),maxcps(point),
     2                      avcps(point),ectyp,conten(point)
         point = list(point)
```

```
        go to 6020
c
 6030 continue
      write (output,628)
      write (output,629)
c
      write (output,711)
      write (output,712)
c
      write (output,721)
      write (output,722)
      write (output,723)
      write (output,724)
      write (output,725)
      write (output,726)
      write (output,727)
c
      point = laststst
 7020 continue
      if (point .eq. none) go to 7030
         do 7100 i=1,3
            name(i) = ptname(point,i)
 7100    continue
         ectyp = sqrt(stdst(point) - (avst(point)*avst(point)))
         write (output,731),name,nbst(point),minst(point),
     1                      maxst(point),avst(point),ectyp
         point = list(point)
         go to 7020
c
 7030 continue
      write (output,728)
      write (output,729)
c
  999 continue
      return
c
c
c
  101 format (1h1,10x,'*********************************************')
  102 format (1x ,10x,'*                                           *')
  103 format (1x ,10x,'*     A:   Stations multiples:  Storages    *')
  104 format (1x ,10x,'*                                           *')
  105 format (1x ,10x,'*********************************************')
c
  151 format (1h-,10x,'  Ce programme n''a pas genere de storages !  ')
  152 format (1x ,10x,'*********************************************')
c
  211 format (1h-,'*  1:   Informations d''ordre general !  ')
  212 format (1x ,'****************************************')
c
  221 format (1h0,'*************************************************'
     1, '*************************************************')
  222 format (1x ,'*                 *            *             *   '
     1 ,'          *            *            *            *')
  223 format (1x ,'* Identification  * Nb entrees  * Nb entrees  *  %'
     1 ,' entrees  * Tps observ  * Tps utilis  * Taux utilis *')
  224 format (1x ,'*                 *            * sans attente* sa'
     1 ,'ns attente* des statist *de la station*de la station*')
  225 format (1x ,'*                 *            *             *   '
```

```
     1 ,'          *             *             *             *')
   226 format (1x ,'*************************************************'
     1 ,'*************************************************')
   227 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *             *')
   231 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,3x,f6.2,
     1       4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,4x,f5.2,4x,1h*)
   228 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *             *')
   229 format (1x ,'*************************************************'
     1, '*************************************************')
c
   311 format (1h1,'*  2:   Statistiques sur le nombre d''unites demand'
     1 ,'ees par transaction !  ')
   312 format (1x ,'*************************************************'
     1 ,'************************')
c
   321 format (1h0,'*************************************************'
     1 ,'************************************')
   322 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
   323 format (1x ,'* Identification  *  Nombre des *   Demande   *   '
     1 ,'Demande   *   Demande   *  Ecart-type *')
   324 format (1x ,'*                    * observations*   minimale  *   '
     1 ,'maximale  *   moyenne   *de la demande*')
   325 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
   326 format (1x ,'*************************************************'
     1 ,'************************************')
   327 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
   331 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,4x,i5,4x,
     1           1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*)
   328 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
   329 format (1x ,'*************************************************'
     1 ,'************************************')
c
   411 format (1h1,'*  3:  Statistiques sur le temps d''attente devant '
     1 ,'le serveur (conditionnelle au fait qu''il y ait effectivement'
     2 ,' eu une attente) !  ')
   412 format (1x ,'*************************************************'
     1 ,'*************************************************'
     2 ,'********************')
c
   421 format (1h0,'*************************************************'
     1 ,'************************************')
   422 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
   423 format (1x ,'* Identification  *  Nombre des *   Attente   *   '
     1 ,'Attente   *   Attente   *  Ecart-type *')
   424 format (1x ,'*                    * observations*   minimale  *   '
     1 ,'maximale  *   moyenne   * de l''attente*')
   425 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
   426 format (1x ,'*************************************************'
     1 ,'************************************')
   427 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
```

```
  431 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,
     1           2x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*)
  428 format (1x ,'*                  *            *            *   '
     1 ,'          *            *            *')
  429 format (1x ,'*********************************************************'
     1 ,'*************************************')
c
  511 format (1h1,'*  4:  Statistiques sur la longueur de la file d''a'
     1 ,'ttente devant le serveur !  ')
  512 format (1x ,'*********************************************************'
     1 ,'****************************')
c
  521 format (1h0,'*********************************************************'
     1 ,'*************************************************')
  522 format (1x ,'*                  *            *            *   '
     1 ,'          *            *            *            *')
  523 format (1x ,'* Identification  *   Longueur  *   Longueur  *   '
     1 ,'Longueur  *   Longueur  * Ecart-type *   Longueur  *')
  524 format (1x ,'*                  *   initiale  *   minimale  *   '
     1 ,'maximale  *   moyenne   * de longueur *   courante  *')
  525 format (1x ,'*                  *             *            *   '
     1 ,'          *            *            *            *')
  526 format (1x ,'*********************************************************'
     1 ,'*************************************************')
  527 format (1x ,'*                  *             *            *   '
     1 ,'          *            *            *            *')
  531 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,4x,i5,4x,
     1           1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,4x,i5,4x,1h*)
  528 format (1x ,'*                  *             *            *   '
     1 ,'          *            *            *            *')
  529 format (1x ,'*********************************************************'
     1 ,'*************************************************')
c
  611 format (1h1,'*  5:   Statistiques sur le nombre d''unites occupe'
     1 ,'es de la station !  ')
  612 format (1x ,'*********************************************************'
     1 ,'********************')
c
  621 format (1h0,'*********************************************************'
     1 ,'*********************************************************'
     2 ,'*******')
  622 format (1x ,'*                  *            *            *   '
     1 ,'          *            *            *            *      '
     2 ,'      *')
  623 format (1x ,'* Identification  *   Capacite  *  Occupation *  O'
     1 ,'ccupation *  Occupation *  Occupation *  Ecart-type *  Occup'
     2 ,'ation *')
  624 format (1x ,'*                  *de la station*   initiale  *   '
     1 ,'minimale  *   maximale  *   moyenne   * d''occupation*   cour'
     2 ,'ante  *')
  625 format (1x ,'*                  *             *            *   '
     1 ,'          *            *            *            *      '
     2 ,'      *')
  626 format (1x ,'*********************************************************'
     1 ,'*********************************************************'
     2 ,'*******')
  627 format (1x ,'*                  *             *            *   '
     1 ,'          *            *            *            *      '
     2 ,'      *')
```

```
  631 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,4x,i5,4x,
     1            1h*,4x,i5,4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,
     2            4x,i5,4x,1h*)
  628 format (1x ,'*                *            *            *   '
     1 ,'         *            *            *            *      '
     2 ,'      *')
  629 format (1x ,'*****************************************************'
     1 ,'*************************************************************'
     2 ,'*******')
c
  711 format (1h1,'*  6:   Statistiques sur le temps de service d''une'
     1 ,' transaction par le serveur !  ')
  712 format (1x ,'*****************************************************'
     1 ,'********************************')
c
  721 format (1h0,'*****************************************************'
     1 ,'*************************************')
  722 format (1x ,'*                *            *            *   '
     1 ,'         *            *            *')
  723 format (1x ,'* Identification  *  Nombre des *   Service   *   '
     1 ,'Service   *   Service   *  Ecart-type *')
  724 format (1x ,'*                 * observations*   minimal   *   '
     1 ,'maximal   *   moyen     *  du service *')
  725 format (1x ,'*                 *             *             *   '
     1 ,'         *            *            *')
  726 format (1x ,'*****************************************************'
     1 ,'*************************************')
  727 format (1x ,'*                 *             *             *   '
     1 ,'         *            *            *')
  731 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,
     1            2x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*)
  728 format (1x ,'*                 *             *             *   '
     1 ,'         *            *            *')
  729 format (1x ,'*****************************************************'
     1 ,'*************************************')
c
c
c
      end
c*****************************************************************
      subroutine facrep
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ class(1),pred(1),suc(1),list(1),occup(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /tmps/ time
      common /io/ input,output
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
      dimension name(3)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq
      real tpus,tpini,lstpus,time
```

```
      real tpsobs,txus,ectyp,txnul
c
      write (output,101)
      write (output,102)
      write (output,103)
      write (output,104)
      write (output,105)
c
      if (lastfc .ne. none) go to 2010
         write (output,151)
         write (output,152)
         go to 999
c
 2010 continue
      write (output,211)
      write (output,212)
c
      write (output,221)
      write (output,222)
      write (output,223)
      write (output,224)
      write (output,225)
      write (output,226)
      write (output,227)
c
      point = lastfc
 2020 continue
      if (point .eq. none) go to 2030
         do 2100 i=1,3
            name(i) = ptname(point,i)
 2100    continue
         if (occup(point) .eq. none) go to 2025
            tpus(point) = tpus(point) + (time - lstpus(point))
            lstpus(point) = time
 2025    continue
         txnul = (float(nulent(point))/float(nbent(point))) * 100.0
         tpsobs = time - tpini(point)
         txus = (tpus(point) / tpsobs) * 100.0
         write (output,231),name,nbent(point),nulent(point),txnul,
     1                      tpsobs,tpus(point),txus
         point = list(point)
         go to 2020
c
 2030 continue
      write (output,228)
      write (output,229)
c
      write (output,411)
      write (output,412)
c
      write (output,421)
      write (output,422)
      write (output,423)
      write (output,424)
      write (output,425)
      write (output,426)
      write (output,427)
c
      point = lastfc
```

```
 4020 continue
      if (point .eq. none) go to 4030
         do 4100 i=1,3
            name(i) = ptname(point,i)
 4100    continue
         ectyp = sqrt(stdwt(point) - (avwt(point)*avwt(point)))
         write (output,431),name,nbwt(point),minwt(point),
     1                      maxwt(point),avwt(point),ectyp
         point = list(point)
         go to 4020
c
 4030 continue
      write (output,428)
      write (output,429)
c
      write (output,511)
      write (output,512)
c
      write (output,521)
      write (output,522)
      write (output,523)
      write (output,524)
      write (output,525)
      write (output,526)
      write (output,527)
c
      point = lastfc
 5020 continue
      if (point .eq. none) go to 5030
         do 5100 i=1,3
            name(i) = ptname(point,i)
 5100    continue
         call addcap (avcpq(point),stdcpq(point),nbcpq(point),
     1                mincpq(point),maxcpq(point),curcpq(point),
     2                time-tpscpq(point))
         tpscpq(point) = time
         ectyp = sqrt(stdcpq(point) - (avcpq(point)*avcpq(point)))
         write (output,531),name,inicpq(point),mincpq(point),
     1                      maxcpq(point),avcpq(point),
     2                      ectyp,curcpq
         point = list(point)
         go to 5020
c
 5030 continue
      write (output,528)
      write (output,529)
c
      write (output,711)
      write (output,712)
c
      write (output,721)
      write (output,722)
      write (output,723)
      write (output,724)
      write (output,725)
      write (output,726)
      write (output,727)
c
      point = lastfc
```

```
 7020 continue
      if (point .eq. none) go to 7030
         do 7100 i=1,3
            name(i) = ptname(point,i)
 7100    continue
         ectyp = sqrt(stdst(point) - (avst(point)*avst(point)))
         write (output,731),name,nbst(point),minst(point),
     1                      maxst(point),avst(point),ectyp
         point = list(point)
         go to 7020
c
 7030 continue
      write (output,728)
      write (output,729)
c
  999 continue
      return
c
c
c
  101 format (1h1,10x,'*********************************************')
  102 format (1x ,10x,'*                                           *')
  103 format (1x ,10x,'*     B:   Stations simples:  Facilities    *')
  104 format (1x ,10x,'*                                           *')
  105 format (1x ,10x,'*********************************************')
c
  151 format (1h-,10x,'  Ce programme n''a pas genere de facilities ! ')
  152 format (1x ,10x,'*********************************************')
c
  211 format (1h-,'*  1:   Informations d''ordre general !  ')
  212 format (1x ,'****************************************')
c
  221 format (1h0,'*****************************************************'
     1, '*****************************************************')
  222 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *')
  223 format (1x ,'* Identification  * Nb entrees  * Nb entrees  *  %'
     1 ,' entrees  * Tps observ  * Tps utilis  * Taux utilis *')
  224 format (1x ,'*                 *             * sans attente* sa'
     1 ,'ns attente* des statist *de la station*de la station*')
  225 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *')
  226 format (1x ,'*****************************************************'
     1 ,'*****************************************************')
  227 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *')
  231 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,3x,f6.2,
     1       4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,4x,f5.2,4x,1h*)
  228 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *')
  229 format (1x ,'*****************************************************'
     1, '*****************************************************')
c
  411 format (1h1,'*  2:  Statistiques sur le temps d''attente devant '
     1 ,'le serveur (conditionnelle au fait qu''il y ait effectivement'
     2 ,' eu une attente) !  ')
  412 format (1x ,'*****************************************************'
     1 ,'*************************************************************'
     2 ,'*********************')
```

```
c
  421 format (1h0,'*************************************************'
     1 ,'*************************************')
  422 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  423 format (1x ,'* Identification  *  Nombre des *   Attente   *   '
     1 ,'Attente   *   Attente   *  Ecart-type *')
  424 format (1x ,'*                  * observations*   minimale  *   '
     1 ,'maximale  *   moyenne   * de l''attente*')
  425 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  426 format (1x ,'*************************************************'
     1 ,'*************************************')
  427 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  431 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,
     1           2x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*)
  428 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  429 format (1x ,'*************************************************'
     1 ,'*************************************')
c
  511 format (1h1,'*  3:  Statistiques sur la longueur de la file d''a'
     1 ,'ttente devant le serveur !  ')
  512 format (1x ,'*************************************************'
     1 ,'*****************************')
c
  521 format (1h0,'*****************************************************'
     1 ,'*****************************************')
  522 format (1x ,'*                  *             *                 *   '
     1 ,'          *             *             *             *')
  523 format (1x ,'* Identification  *  Longueur   *   Longueur    *   '
     1 ,'Longueur  *   Longueur  *  Ecart-type *   Longueur  *')
  524 format (1x ,'*                  *   initiale  *   minimale    *   '
     1 ,'maximale  *   moyenne   * de longueur *   courante  *')
  525 format (1x ,'*                  *             *                 *   '
     1 ,'          *             *             *             *')
  526 format (1x ,'*****************************************************'
     1 ,'*****************************************')
  527 format (1x ,'*                  *             *                 *   '
     1 ,'          *             *             *             *')
  531 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,4x,i5,4x,
     1           1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,4x,i5,4x,1h*)
  528 format (1x ,'*                  *             *                 *   '
     1 ,'          *             *             *             *')
  529 format (1x ,'*****************************************************'
     1 ,'*****************************************')
c
  711 format (1h1,'*  4:   Statistiques sur le temps de service d''une'
     1 ,' transaction par le serveur !  ')
  712 format (1x ,'*************************************************'
     1 ,'*******************************')
c
  721 format (1h0,'*************************************************'
     1 ,'*************************************')
  722 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  723 format (1x ,'* Identification  *  Nombre des *   Service   *   '
     1 ,'Service   *   Service   *  Ecart-type *')
```

```
  724 format (1x ,'*                  * observations*   minimal   *   '
     1 ,'maximal   *   moyen     *  du service *')
  725 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  726 format (1x ,'*************************************************'
     1 ,'*************************************')
  727 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  731 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,
     1           2x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*)
  728 format (1x ,'*                  *             *             *   '
     1 ,'          *             *             *')
  729 format (1x ,'*************************************************'
     1 ,'*************************************')
c
c
c
      end
c*************************************************************
      subroutine regrep (reg)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /tmps/ time
      common /io/ input,output
      common /objatr/ class(1),pred(1),suc(1),list(1),curct(1),hidon(1)
      common /objatr/ ptname(1,3),nbent(1),nbout(1),tpini(1)
      common /objatr/ mintt(1),maxtt(1),avtt(1),stdtt(1),nbtt(1)
      common /objatr/ inict(1),minct(1),maxct(1),avct(1),stdct(1)
      common /objatr/ tpsct(1),nbtct(1)
      common /gpsss/ storag,facily,groupe,region,histo
      common /lst/ laststt,lastfc,lastgr,lastrg,lasths
      real time,tpini,tpsct,nbtct
      real mintt,maxtt,avtt,stdtt,avct,stdct
      real tpsobs,ectyp
      dimension name(3)
c
      write (output,101)
      write (output,102)
      write (output,103)
      write (output,104)
      write (output,105)
c
      if (lastrg .ne. none) go to 2010
           write (output,151)
           write (output,152)
           go to 999
c
 2010 continue
      write (output,211)
      write (output,212)
c
      write (output,221)
      write (output,222)
      write (output,223)
      write (output,224)
      write (output,225)
      write (output,226)
      write (output,227)
```

```
c
      point = lastrg
 2020 continue
      if (point .eq. none) go to 2030
           do 2100 i=1,3
                name(i) = ptname(point,i)
 2100      continue
           tpsobs = time - tpini(point)
           ectyp = sqrt (stdtt(point) - (avtt(point)*avtt(point)))
           write (output,231),name,tpsobs,nbent(point),nbout(point),
     1                      mintt(point),maxtt(point),avtt(point),
     2                      ectyp
           point = list(point)
           go to 2020
c
 2030 continue
      write (output,228)
      write (output,229)
c
      write (output,411)
      write (output,412)
c
      write (output,421)
      write (output,422)
      write (output,423)
      write (output,424)
      write (output,425)
      write (output,426)
      write (output,427)
c
      point = lastrg
 4020 continue
      if (point .eq. none) go to 4030
           do 4100 i=1,3
                name(i) = ptname(point,i)
 4100      continue
           call addcap (avct(point),stdct(point),nbtct(point),
     1                 minct(point),maxct(point),curct(point),
     2                 time-tpsct(point))
           tpsct(point) = time
           ectyp = sqrt (stdct(point) - (avct(point)*avct(point)))
           write (output,431),name,inict(point),minct(point),
     1                      maxct(point),avct(point),ectyp,
     2                      curct(point)
           point = list(point)
           go to 4020
c
 4030 continue
      write (output,428)
      write (output,429)
c
  999 continue
      return
c
c
c
  101 format (1h1,10x,'*******************************************')
  102 format (1x ,10x,'*                                         *')
  103 format (1x ,10x,'*     C:   Regions                        *')
```

```
  104 format (1x ,10x,'*                                              *')
  105 format (1x ,10x,'**********************************************')
c
  151 format (1h-,10x,'  Ce programme n''a pas genere de regions ! ')
  152 format (1x ,10x,'**********************************************')
c
  211 format (1h-,'*  1:   Statistiques sur le temps de passage de ch'
     1 ,'acune des transactions a travers la region !  ')
  212 format (1x ,'************************************************'
     1 ,'******************************************')
c
  221 format (1h0,'************************************************'
     1 ,'************************************************************'
     2 ,'*******')
  222 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *      '
     2 ,'      *')
  223 format (1x ,'* Identification  * Tps observ  * Nb entrees  * Nb'
     1 ,' sorties  * Tps passage * Tps passage * Tps passage *  Ec-ty'
     2 ,'pe du *')
  224 format (1x ,'*                 * des statist *             *   '
     1 ,'          *    minimal  *   maximal   *    moyen    * tps pa'
     2 ,'ssage *')
  225 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *      '
     2 ,'      *')
  226 format (1x ,'************************************************'
     1 ,'************************************************************'
     2 ,'*******')
  227 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *      '
     2 ,'      *')
  231 format (1x ,1h*,1x,3a5,1x,1h*,2x,f9.3,2x,1h*,4x,i5,4x,1h*,4x,i5,
     1       4x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,2x,f9.3,2x,1h*,2x,
     2       f9.3,2x,1h*)
  228 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *      '
     2 ,'      *')
  229 format (1x ,'************************************************'
     1 ,'************************************************************'
     2 ,'*******')
c
  411 format (1h1,'*  2:  Statistiques sur le nombre de transactions '
     1 ,'presentes dans la region !  ')
  412 format (1x ,'************************************************'
     1 ,'****************************')
c
  421 format (1h0,'************************************************'
     1 ,'************************************************')
  422 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *')
  423 format (1x ,'* Identification  *  Occupation *  Occupation *  O'
     1 ,'ccupation *  Occupation *  Ec-type de *  Occupation *')
  424 format (1x ,'*                 *   initiale  *   minimale  *   '
     1 ,'maximale  *   moyenne   * 1''occupation*   courante  *')
  425 format (1x ,'*                 *             *             *   '
     1 ,'          *             *             *             *')
  426 format (1x ,'************************************************'
     1 ,'************************************************')
```

```
  427 format (1x ,'*                  *            *             *   '
     1 ,'          *             *            *             *')
  431 format (1x ,1h*,1x,3a5,1x,1h*,4x,i5,4x,1h*,4x,i5,4x,1h*,4x,i5,4x,
     1        1h*,2x,f9.2,2x,1h*,2x,f9.2,2x,1h*,4x,i5,4x,1h*)
  428 format (1x ,'*                  *            *             *   '
     1 ,'          *             *            *             *')
  429 format (1x ,'*****************************************************'
     1 ,'*****************************************************')
c
c
c
      end
c**********************************************************************
      subroutine inireg (reg)
c
      implicit integer (a-z)
      common /tmps/ time
      common /objatr/ class(1),pred(1),suc(1),list(1),curct(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nbout(1),tpini(1)
      common /objatr/ mintt(1),maxtt(1),avtt(1),stdtt(1),nbtt(1)
      common /objatr/ inict(1),minct(1),maxct(1),avct(1),stdct(1)
      common /objatr/ tpsct(1),nbtct(1)
      common /gpsss/ storag,facily,groupe,region,histo
      real time,ptstat,tpini,tpsct,nbtct
      real mintt,maxtt,avtt,stdtt,avct,stdct
c
      if (class(reg) .eq. region) go to 100
           call eerror (950)
           go to 999
  100 continue
      nbent(reg) = 0
      nbout(reg) = 0
      tpini(reg) = time
      mintt(reg) = 100000.0
      maxtt(reg) = 0.0
      avtt(reg) = 0.0
      stdtt(reg) = 0.0
      nbtt(reg) = 0
      inict(reg) = curct(reg)
      minct(reg) = "777777777
      maxct(reg) = 0
      avct(reg) = 0.0
      stdct(reg) = 0.0
      tpsct(reg) = time
      nbtct(reg) = 0.0
c
  999 continue
      return
      end
c**********************************************************************
      subroutine inifac (fac)
c
      implicit integer (a-z)
      common /tmps/ time
      common /objatr/ class(1),pred(1),suc(1),list(1),occup(1),bidon(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
```

```
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq
      real tpus,tpini,lstpus
      real time
      dimension pt(1),lsc(1),prior(1),ptstat(1)
      common /gpsss/ storag,facily,groupe,region,histo
c
      if (class(fac) .eq. facily) go to 100
           call eerror (951)
           go to 999
  100 continue
      nbent(fac) = 0
      nulent(fac) = 0
      minwt(fac) = 100000.0
      maxwt(fac) = 0.0
      avwt(fac) = 0.0
      stdwt(fac) = 0.0
      nbwt(fac) = 0
      inicpq(fac) = curcpq(fac)
      mincpq(fac) = "777777777
      maxcpq(fac) = 0
      avcpq(fac) = 0.0
      stdcpq(fac) = 0.0
      nbcpq(fac) = 0.0
      tpscpq(fac) = time
      minst(fac) = 100000.0
      maxst(fac) = 0.0
      avst(fac) = 0.0
      stdst(fac) = 0.0
      nbst(fac) = 0
      tpus(fac) = 0.0
      tpini(fac) = time
      lstpus(fac) = time
c
  999 continue
      return
      end
c*********************************************************************
      subroutine inisto (sto)
c
      implicit integer (a-z)
      common /tmps/ time
      common /objatr/ class(1),pred(1),suc(1),list(1),cap(1),conten(1)
      common /objatr/ ptname(1,3),nbent(1),nulent(1)
      common /objatr/ tpus(1),tpini(1),lstpus(1)
      common /objatr/ minwt(1),maxwt(1),avwt(1),stdwt(1),nbwt(1)
      common /objatr/ inicpq(1),mincpq(1),maxcpq(1),avcpq(1),stdcpq(1)
      common /objatr/ nbcpq(1),tpscpq(1),curcpq(1)
      common /objatr/ minst(1),maxst(1),avst(1),stdst(1),nbst(1)
      common /objatr/ inicps(1),mincps(1),maxcps(1),avcps(1),stdcps(1)
      common /objatr/ nbcps(1),tpscps(1)
      common /objatr/ minrqs(1),maxrqs(1),avrqs(1),stdrqs(1),nbrqs(1)
      real minwt,maxwt,avwt,stdwt,minst,maxst,avst,stdst
      real avcpq,stdcpq,tpscpq,nbcpq,avcps,stdcps,tpscps,nbcps
      real avrqs,stdrqs,tpus,tpini,lstpus,time
c
      if (class(sto) .eq. storag) go to 100
```

```
            call eerror (953)
            go to 999
  100 continue
      nbent(sto) = 0
      nulent(sto) = 0
      minwt(sto) = 100000.0
      maxwt(sto) = 0.0
      avwt(sto) = 0.0
      stdwt(sto) = 0.0
      nbwt(sto) = 0
      inicpq(sto) = curcpq(sto)
      mincpq(sto) = "777777777
      maxcpq(sto) = 0
      avcpq(sto) = 0.0
      stdcpq(sto) = 0.0
      nbcpq(sto) = 0.0
      tpscpq(sto) = time
      minst(sto) = 10000.0
      maxst(sto) = 0.0
      avst(sto) = 0.0
      stdst(sto) = 0.0
      nbst(sto) = 0
      inicps(sto) = conten(sto)
      mincps(sto) ="777777777
      maxcps(sto) = 0
      avcps(sto) = 0.0
      stdcps(sto) = 0.0
      nbcps(sto) = 0.0
      tpscps(sto) = time
      minrqs(sto) = "777777777
      maxrqs(sto) = 0
      avrqs(sto) = 0.0
      stdrqs(sto) = 0.0
      nbrqs(sto) = 0
      tpus(sto) = 0.0
      tpini(sto) = time
      lstpus(sto) = time
c
  999 continue
      return
      end
c*********************************************************************
      subroutine genini
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ bidon(3),list(1)
      common /lst/ laststst,lastfc,lastgr,lastrg,lasths
c
      st = laststst
  100 continue
      if (st .eq. none) go to 200
            call inisto(st)
            st = list(st)
            go to 100
c
  200 continue
      fc = lastfc
  300 continue
```

```
      if (fc .eq. none) go to 400
            call inifac(fc)
            fc = list(fc)
            go to 300
  400 continue
      rg = lastrg
  500 continue
      if (rg .eq. none) go to 999
            call inireg(rg)
            rg = list(rg)
            go to 500
c
  999 continue
      return
      end
c*********************************************************************
      subroutine genrep
c
      implicit integer (a-z)
      common /io/ input,output
c
      write (output,100)
      write (output,101)
      write (output,102)
      write (output,103)
      write (output,104)
      write (output,105)
      write (output,106)
      write (output,107)
      write (output,108)
      write (output,109)
      write (output,110)
      write (output,111)
      write (output,112)
c
      write (output,151)
      write (output,152)
      write (output,153)
c
c
c
  100 format (1h1,30x,'                                              ')
  101 format (1h-,30x,'**********************************************')
  102 format (1x ,30x,'*                                            *')
  103 format (1x ,30x,'*                                            *')
  104 format (1x ,30x,'*                                            *')
  105 format (1x ,30x,'*           S I M U F O R                    *')
  106 format (1x ,30x,'*         ***************************        *')
  107 format (1x ,30x,'*                                            *')
  108 format (1x ,30x,'*                version 1.0                 *')
  109 format (1x ,30x,'*                                            *')
  110 format (1x ,30x,'*                                            *')
  111 format (1x ,30x,'*                                            *')
  112 format (1x ,30x,'**********************************************')
c
  151 format (1h-,30x,'    R A P P O R T     S T A T I S T I Q U E .  ')
  152 format (1x ,30x,'                                              ')
  153 format (1x ,30x,'**********************************************')
c
```

```
c
c
      call storep
      call facrep
      call regrep
      call hisrep
c
  999 continue
      return
      end
```

2. SOUS-PROGRAMMES ET FONCTIONS ASSEMBLEUR.

```
title assem
entry locf,jumpto,sisave,repere,net
locf:   hrrz    0,@16
        popj    17,0
sisave: pop     17,1
        pop     17,2
        movem   2,@0(16)
        push    17,1
        popj    17,0
jumpto: pop     17,1
        pop     17,1
        push    17,@0(16)
        popj    17,0
repere: popj    17,0
net:    pop     17,1
        pop     17,2
        push    17,1
        popj    17,0
        end
```

3. EXERCICE DE LA TAVERNE. (o)

3.1. ENONCE.

La taverne que je fréquente possède une grande salle, capable de contenir 100 personnes. Elle ouvre ses portes à 17 h. et ferme à 3 h. du matin.

Les clients arrivent en groupes. S'il y a assez de places dans la salle pour tous les membres du groupe, ils s'installent dans la taverne; sinon, le groupe s'en va ailleurs.

Des groupes arrivent au hasard, l'intervalle moyen entre les groupes étant le suivant:

- de 17 h. à 20 h. - 4 minutes,
- de 20 h. à 23 h. - 2 minutes,
- de 23 h. à 01 h. - 4 minutes.

Un groupe inclut de 1 à 8 personnes, avec les probabilités données ci-après:

nombre dans le groupe	1	2	3	4	6	8
probabilités	0.1	0.45	0.04	0.3	0.07	0.04

Quand un groupe s'est installé dans la taverne, chaque membre du groupe commande une bière. Au bout d'une certaine période P, chaque membre du groupe a bu sa bière, et le groupe décide, soit de rester dans la taverne, soit de s'en aller. (Tous le membres du groupe partent ensemble.) La décision de rester est prise avec une probabilité de 0.6, et la décision de partir avec une probabilité de 0.4. Si le groupe décide de rester, chaque membre commande une autre bière. Une autre période P sera nécessaire pour boire cette bière, et ensuite le groupe doit encore une fois décider soit de rester, soit de partir. Cette nouvelle décision est prise avec les mêmes probabilités et les mêmes conséquences qu'auparavant, et ainsi de suite.

Les périodes P sont indépendantes, et suivent une distribution uniforme entre 10.0 et 50.0 minutes.

La situation est compliquée par le fait que la taverne présente des stripteaseuses, de 18 h. jusqu'à la fermeture. La première danseuse commence à 18 h. exactement, et son numéro dure 10 +ou- 5 minutes. Le numéro est suivi d'une pause de 20 +ou- 5 minutes. Ensuite la deuxième stripteaseuse commence, danse pendant 10 +ou- 5 minutes, il y a une pause de 20 +ou- 5 minutes, etc. A 2 h., le striptease en cours se termine normalement (s'il y en a un), et ensuite on ne danse plus.

Durant le striptease, les lumières de la salle sont éteintes, et on ne peut ni sortir ni entrer. Donc les groupes qui décident de partir sont obligés d'attendre la fin du numéro avant de quitter la taverne, et les groupes qui arrivent sont aussi obligés d'attendre pour voir s'il y a des places dans la salle. Evidemment un groupe qui arrive peut prendre des places libérées par un autre groupe qui s'en va exactement au même moment.

(°) D'après:
Exercice de programmation de simulation.
ECOLE INTERNATIONALE
D'ETE D'INFORMATIQUE
A.F.C.E.T. - Session 1977
Montréal, 18-30 juillet.

3.2. PROGRAMME EN SIMUFOR.

```
      program stript
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /objatr/ bidon(10)
      common /global/ moy,spect,fred,inq,outq
      common /result/ table1,nbcli,table2,nbspec
      real moy,time
      dimension table1 (500,8),table2 (30,5)
      logical spect
      external eff,arrive,porte
c
      call init
c   initialisation des tables resultat
      call maz ( table1,500,8)
      call maz ( table2,30,5)
      spect=.false.
c  generation de l'attraction
      loc1=locf(porte)
      employ=class(loc1,9)
      fred=new(employ)
c
      inq=new(head)
      outq=new(head)
c
      loc2=locf(eff)
      gogo=class(loc2,7)
      girl=new(gogo)
      call activ(girl,delay,60.0)
c  generation des spectateurs
      loc3=locf(arrive)
      arriv=class(loc3,9)
      il=new(arriv)
      moy=4.0
      call activ(il,delay,0.0)
      call hold(180.0)
      moy=2.0
      call hold(180.0)
      moy=4.0
      call hold(120.0)
c  fin du spectacle
      call cancel(il)
      call hold(120.0)
      call prtab1
      call prtab2
      call repere
      stop
      end
c
c
c
      subroutine eff
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
```

```
      common /objatr/ bidon(7)
      common /local1/ a,u1,u2
      common /global/ moy,spect,fred,inq,outq
      common /result/ table1,nbcli,table2,nbspec
      logical spect
      dimension table1(500,8), table2(30,5)
      real unif,time
c
      nbspec = 0
      u1=153251
      u2=172543
      a=1
  100 continue
      a=mod(a+1,2)
      spect=.true.
      nbspec=nbspec + 1
      aa=a+1
      call hour(hr,mn)
      table2(nbspec,1) = aa
      table2(nbspec,2) = hr
      table2(nbspec,3) = mn
      call hold(unif(5.0,15.0,u1))
      spect=.false.
      call hour(hr,mn)
      table2(nbspec,4) = hr
      table2(nbspec,5) = mn
      call activ(fred,delay,0.0)
      call hold(unif(15.0,25.0,u2))
      call temps (time)
      if (time.lt.480.0) goto 100
      call endpro
c
  999 continue
      return
      end
c
c
c
      subroutine group
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /objatr/ bidon(7),num(1),nb(1)
      common /global/ moym,spect,fred,inq,outq
      common /local2/ u1,u2
      common /result/ table1,nbcli,table2,nbspec
      real p,unif
      real time
      dimension table1 (500,8),table2 (30,5)
      logical spect,draw
c
      u1=145333
      u2=991111
      call hour(hr,mn)
      call into(actuel,inq)
      table1(num(actuel),2)=hr
      table1(num(actuel),3)=mn
      if (spect) goto 100
```

```
      call activ(fred,delay,0.0)
      call passiv
      goto 200
c
  100 continue
      call passiv
c
  200 continue
      call hour(hr,mn)
      table1(num(actuel),4)=hr
      table1(num(actuel),5)=mn
  225 continue
      table1(num(actuel),6)=table1(num(actuel),6) + 1
      p=unif(10.0,50.0,u1)
      call hold(p)
      if ( .not. draw(0.6,u2)) goto 250
      call temps (time)
      if (time .lt. 540.0) goto 225
  250 continue
      call into(actuel,outq)
      if (spect) goto 300
      call activ(fred,delay,0.0)
      call passiv
      goto 999
c
  300 continue
      call passiv
c
  999 continue
      call hour(hr,mn)
      table1(num(actuel),7)=hr
      table1(num(actuel),8)=mn
      call endpro
      return
      end
c
c
c
      subroutine porte
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /objatr/ bidon(7),num(1),nb(1)
      common /global/ moy,spect,fred,inq,outq
      common /local3/ nbpla,gr
      common /result/ table1,nbcli,table2,nbspec
      logical empty
      dimension table1 (500,8),table2 (30,5)
      real time
c
      nbpla=100
c
  100 continue
      if (empty(outq)) goto 200
      gr=first(outq)
      call out(gr)
      nbpla=nbpla+nb(gr)
      call activ(gr,delay,0.0)
```

```
      goto 100
c
  200 continue
      if (empty(inq)) goto 300
      gr=first(inq)
      call out(gr)
      if (nbpla.lt.nb(gr)) goto 250
      nbpla=nbpla-nb(gr)
      call activ(gr,delay,0.0)
      goto 200
c
  250 continue
      call hour (hr,mn)
      table1 (num(gr),7) = hr
      table1 (num(gr),8) = mn
      call return(gr)
      goto 200
c
  300 continue
      call passiv
      goto 100
c
  999 continue
      return
      end
c
c
c
      subroutine arrive
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /global/ moy,spect,fred,inq,outq
      common /objatr/ bidon(7),num(1),nb(1)
      common /local4/ u1,u2,groupp,gr,i1
      common /result/ table1,nbcli,table2,nbspec
      real negexp,rand,moy,i1,time
      dimension table1 (500,8),table2 (30,5)
      external group
c
      nbcli=1
      u1=355555
      u2=133597
      loc4=locf(group)
      groupp=class(loc4,9)
c
  100 continue
      call hold(negexp(1/moy,u1))
      call temps (time)
      if (time.gt.600.0) goto 999
      gr=new(groupp)
      i1=rand(u2)
      if (i1.ge.0.0.and.i1.lt.0.1) nb(gr)=1
      if (i1.ge.0.1.and.i1.lt.0.55) nb(gr)=2
      if (i1.ge.0.55.and.i1.lt.0.59) nb(gr)=3
      if (i1.ge.0.59.and.i1.lt.0.89) nb(gr)=4
      if (i1.ge.0.89.and.i1.lt.0.96) nb(gr)=6
      if (i1.ge.0.96.and.i1.lt.1.0) nb(gr)=8
```

```
      num(gr)=nbcli
      table1(num(gr),1)=nb(gr)
      call activ(gr,delay,0.0)
      nbcli=nbcli+1
      goto 100
c
  999 continue
      call endpro
      return
      end
c
c
c
      subroutine hour(hr,mn)
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      real time
c
      call temps (time)
      h=int(time/60.0)
      mn=int(time-h*60.0)
      hr=mod((h+17),24)
c
  999 continue
      return
      end
c
c
c
      subroutine prtabl
      implicit integer (a-z)
      common /result/ table1,nbcli,table2,nbspec
      common /io/ input,output
      dimension table1 (500,8),table2 (30,5)
      real temp,tempp
      do 800 i3=1,nbcli-1
                templ=ifix ((float(i3-1))/30.0)
                tempp = (float(i3-1))/30.0
                temp = tempp - float(templ)
                if (temp .gt. 0.0001) goto 200
                      write ( output, 101)
                      write ( output, 102 )
                      write ( output, 103 )
                      write ( output, 109 )
  200           continue
                if (table1(i3,6) .eq. 0) goto 300
                   write ( output, 250 ),i3,table1(i3,1),table1(i3,2)
     1                        ,table1(i3,3),table1(i3,4),table1(i3,5)
     2                        ,table1(i3,6),table1(i3,7),table1(i3,8)
                   goto 400
  300           continue
                   write ( output, 260 ),i3,table1(i3,1),table1(i3,2)
     1                        ,table1(i3,3),table1(i3,6)
     2                        ,table1(i3,7),table1(i3,8)
  400           continue
                templ=ifix ((float(i3))/30.0)
                tempp = (float(i3))/30.0
```

```
                temp = tempp - float(templ)
                if (temp .gt. 0.0001) goto 800
                   write ( output, 109 )
  800           continue
      write ( output, 109 )
  101 format (1h1,/////,' ******************************************'
     1           ,'**************************************')
  102 format (1x,'* Groupe  * Nombre  *    Heure  d"   *    Heure  d" '
     1           ,'  * Nombre de *    Heure de    *')
  103 format (1x,'* Numero  * Clients *    Arrivee     *    Entree    '
     1           ,'  * Tournees  *    Sortie      *')
  250   format ( 1x, '*   ',i3,'   *   ',i3,
     1              '   * ',i2,' Hr , ',i2,' Min * ',
     2                   i2,' Hr , ',i2,' Min *     ',
     3              i2,'    * ',
     4                   i2,' Hr , ',i2,' Min *')
  260   format ( 1x, '*   ',i3,'   *   ',i3,
     1              '   * ',i2,' Hr , ',i2,' Min * '
     2                   ,'    no entry   *     ',
     3              i2,'    * ',
     4                   i2,' Hr , ',i2,' Min *')
  109 format (1x,'******************************************************'
     1           ,'********************************')
c
      return
      end
c
c
      subroutine prtab2
      implicit integer (a-z)
      common /result/ table1,nbcli,table2,nbspec
      common /io/ input,output
      dimension table1 (500,8),table2 (30,5)
c
      write ( output, 100 )
      write ( output, 101 )
      write ( output, 102 )
      write ( output, 109 )
c
      do 300 i=1,nbspec-1
        write ( output, 200 ),i,table2(i,1),table2(i,2),table2(i,3)
     1          ,table2(i,4),table2(i,5)
  300   continue
      write ( output, 109 )
  100 format (1h1,/////,'******************************'
     1           ,'********************************')
  101 format (1x,'* Numero du * Numero de la   *   Heure de  '
     1           ,'*   Heure de   *')
  102 format (1x,'* Spectacle * Stripteaseuse  *   Debut     *'
     1           ,'   Fin        *')
  109 format (1x,'******************************'
     1           ,'********************************')
  200 format (1x,'*     ',i2,'    *        ',i1,'       * ',
     1           i2,' Hr ',i2,' Mn * ',i2,' Hr ',i2,' Mn  *')
      return
      end
c
c
      subroutine maz ( nomtab,dim1,dim2 )
```

```
      implicit integer (a-z)
      dimension nomtab (diml,dim2)
c
      do 100 i = 1,diml
             do 200 j = 1,dim2
                    nomtab(i,j) = 0
  200        continue
  100 continue
      return
      end
```

3.3. RESULTATS.

Groupe Numero	Nombre Clients	Heure d" Arrivee	Heure d" Entree	Nombre de Tournees	Heure de Sortie
1	2	17 Hr , 3 Min	17 Hr , 3 Min	6	19 Hr , 42 Min
2	3	17 Hr , 7 Min	17 Hr , 7 Min	4	19 Hr , 14 Min
3	4	17 Hr , 14 Min	17 Hr , 14 Min	1	17 Hr , 53 Min
4	1	17 Hr , 17 Min	17 Hr , 17 Min	1	17 Hr , 56 Min
5	4	17 Hr , 17 Min	17 Hr , 17 Min	9	21 Hr , 7 Min
6	4	17 Hr , 23 Min	17 Hr , 23 Min	2	18 Hr , 39 Min
7	4	17 Hr , 23 Min	17 Hr , 23 Min	1	18 Hr , 12 Min
8	4	17 Hr , 25 Min	17 Hr , 25 Min	1	18 Hr , 12 Min
9	2	17 Hr , 32 Min	17 Hr , 32 Min	4	19 Hr , 33 Min
10	2	17 Hr , 39 Min	17 Hr , 39 Min	2	18 Hr , 36 Min
11	4	17 Hr , 44 Min	17 Hr , 44 Min	1	18 Hr , 24 Min
12	2	18 Hr , 8 Min	18 Hr , 12 Min	2	19 Hr , 9 Min
13	2	18 Hr , 12 Min	18 Hr , 12 Min	2	19 Hr , 9 Min
14	2	18 Hr , 19 Min	18 Hr , 19 Min	2	19 Hr , 16 Min
15	2	18 Hr , 20 Min	18 Hr , 20 Min	1	18 Hr , 59 Min
16	4	18 Hr , 28 Min	18 Hr , 35 Min	1	18 Hr , 52 Min
17	1	18 Hr , 30 Min	18 Hr , 35 Min	1	19 Hr , 5 Min
18	2	18 Hr , 31 Min	18 Hr , 35 Min	6	21 Hr , 33 Min
19	2	18 Hr , 34 Min	18 Hr , 35 Min	3	20 Hr , 9 Min
20	2	18 Hr , 45 Min	18 Hr , 45 Min	3	19 Hr , 59 Min
21	2	18 Hr , 47 Min	18 Hr , 47 Min	1	19 Hr , 33 Min
22	2	18 Hr , 51 Min	18 Hr , 51 Min	1	19 Hr , 33 Min
23	4	18 Hr , 51 Min	18 Hr , 51 Min	1	19 Hr , 33 Min
24	2	18 Hr , 58 Min	18 Hr , 58 Min	4	20 Hr , 52 Min
25	4	18 Hr , 59 Min	18 Hr , 59 Min	1	19 Hr , 39 Min
26	2	19 Hr , 8 Min	19 Hr , 8 Min	1	19 Hr , 48 Min
27	4	19 Hr , 11 Min	19 Hr , 11 Min	3	21 Hr , 6 Min
28	3	19 Hr , 23 Min	19 Hr , 23 Min	3	20 Hr , 37 Min
29	2	19 Hr , 29 Min	19 Hr , 33 Min	2	20 Hr , 16 Min
30	3	19 Hr , 35 Min	19 Hr , 35 Min	2	20 Hr , 32 Min

Groupe Numero	Nombre Clients	Heure d" Arrivee	Heure d" Entree	Nombre de Tournees	Heure de Sortie
31	2	19 Hr , 37 Min	19 Hr , 37 Min	6	22 Hr , 55 Min
32	2	19 Hr , 38 Min	19 Hr , 38 Min	2	21 Hr , 6 Min
33	8	19 Hr , 41 Min	19 Hr , 41 Min	1	20 Hr , 21 Min
34	2	19 Hr , 51 Min	19 Hr , 51 Min	3	21 Hr , 14 Min
35	4	19 Hr , 54 Min	19 Hr , 59 Min	1	20 Hr , 16 Min
36	4	20 Hr , 0 Min	20 Hr , 0 Min	1	20 Hr , 40 Min
37	2	20 Hr , 1 Min	20 Hr , 1 Min	1	20 Hr , 41 Min
38	2	20 Hr , 2 Min	20 Hr , 2 Min	2	21 Hr , 7 Min
39	2	20 Hr , 5 Min	20 Hr , 5 Min	2	21 Hr , 6 Min
40	4	20 Hr , 5 Min	20 Hr , 5 Min	1	20 Hr , 45 Min
41	4	20 Hr , 9 Min	20 Hr , 9 Min	5	23 Hr , 3 Min
42	2	20 Hr , 10 Min	20 Hr , 10 Min	1	20 Hr , 50 Min
43	2	20 Hr , 11 Min	20 Hr , 11 Min	4	22 Hr , 2 Min
44	3	20 Hr , 14 Min	20 Hr , 14 Min	4	21 Hr , 55 Min
45	2	20 Hr , 19 Min	20 Hr , 19 Min	2	21 Hr , 38 Min
46	4	20 Hr , 22 Min	20 Hr , 22 Min	4	22 Hr , 38 Min
47	2	20 Hr , 23 Min	20 Hr , 30 Min	1	20 Hr , 47 Min
48	3	20 Hr , 24 Min	20 Hr , 30 Min	1	21 Hr , 6 Min
49	4	20 Hr , 27 Min	20 Hr , 30 Min	1	21 Hr , 15 Min
50	4	20 Hr , 30 Min	20 Hr , 30 Min	4	22 Hr , 37 Min
51	4	20 Hr , 34 Min	20 Hr , 34 Min	4	22 Hr , 5 Min
52	4	20 Hr , 35 Min	20 Hr , 36 Min	4	22 Hr , 25 Min
53	4	20 Hr , 39 Min	20 Hr , 39 Min	2	21 Hr , 36 Min
54	2	20 Hr , 45 Min	20 Hr , 45 Min	1	21 Hr , 33 Min
55	2	20 Hr , 49 Min	20 Hr , 49 Min	1	21 Hr , 33 Min
56	8	20 Hr , 50 Min	20 Hr , 50 Min	1	21 Hr , 33 Min
57	2	20 Hr , 53 Min	20 Hr , 53 Min	2	22 Hr , 12 Min
58	6	20 Hr , 58 Min	21 Hr , 6 Min	6	23 Hr , 43 Min
59	1	21 Hr , 0 Min	21 Hr , 6 Min	2	21 Hr , 40 Min
60	2	21 Hr , 3 Min	21 Hr , 6 Min	4	22 Hr , 46 Min

Groupe Numero	Nombre Clients	Heure d' Arrivee	Heure d' Entree	Nombre de Tournees	Heure de Sortie
61	1	21 Hr , 12 Min	21 Hr , 12 Min	6	0 Hr , 23 Min
62	8	21 Hr , 19 Min	21 Hr , 19 Min	1	21 Hr , 59 Min
63	4	21 Hr , 19 Min	21 Hr , 19 Min	1	21 Hr , 59 Min
64	6	21 Hr , 19 Min	21 Hr , 19 Min	1	21 Hr , 59 Min
65	4	21 Hr , 19 Min	21 Hr , 19 Min	1	21 Hr , 59 Min
66	2	21 Hr , 21 Min	21 Hr , 21 Min	5	0 Hr , 9 Min
67	3	21 Hr , 22 Min	21 Hr , 22 Min	4	23 Hr , 41 Min
68	8	21 Hr , 28 Min	21 Hr , 33 Min	1	21 Hr , 55 Min
69	4	21 Hr , 31 Min	21 Hr , 33 Min	3	22 Hr , 42 Min
70	4	21 Hr , 32 Min	21 Hr , 33 Min	1	22 Hr , 18 Min
71	2	21 Hr , 38 Min	21 Hr , 38 Min	3	23 Hr , 25 Min
72	4	21 Hr , 40 Min	21 Hr , 40 Min	4	0 Hr , 7 Min
73	2	21 Hr , 41 Min	21 Hr , 41 Min	1	22 Hr , 25 Min
74	1	21 Hr , 42 Min	21 Hr , 42 Min	1	22 Hr , 25 Min
75	4	21 Hr , 42 Min	21 Hr , 42 Min	5	23 Hr , 57 Min
76	2	21 Hr , 43 Min	21 Hr , 43 Min	2	22 Hr , 49 Min
77	2	21 Hr , 45 Min	21 Hr , 45 Min	3	23 Hr , 41 Min
78	2	21 Hr , 45 Min	21 Hr , 45 Min	1	22 Hr , 25 Min
79	4	21 Hr , 53 Min	21 Hr , 55 Min	3	23 Hr , 32 Min
80	4	21 Hr , 55 Min	21 Hr , 55 Min	1	22 Hr , 13 Min
81	4	21 Hr , 56 Min	no entry	0	21 Hr , 56 Min
82	4	21 Hr , 58 Min	no entry	0	21 Hr , 58 Min
83	4	21 Hr , 59 Min	21 Hr , 59 Min	2	23 Hr , 4 Min
84	4	22 Hr , 6 Min	22 Hr , 6 Min	1	22 Hr , 46 Min
85	2	22 Hr , 8 Min	22 Hr , 8 Min	2	23 Hr , 28 Min
86	2	22 Hr , 10 Min	22 Hr , 10 Min	1	22 Hr , 55 Min
87	8	22 Hr , 19 Min	22 Hr , 25 Min	5	1 Hr , 1 Min
88	6	22 Hr , 26 Min	22 Hr , 26 Min	2	23 Hr , 25 Min
89	2	22 Hr , 26 Min	22 Hr , 26 Min	4	0 Hr , 23 Min
90	4	22 Hr , 27 Min	22 Hr , 27 Min	2	23 Hr , 54 Min

Groupe Numero	Nombre Clients	Heure d' Arrivee	Heure d' Entree	Nombre de Tournees	Heure de Sortie
91	1	22 Hr , 27 Min	22 Hr , 27 Min	1	23 Hr , 7 Min
92	4	22 Hr , 32 Min	22 Hr , 32 Min	1	23 Hr , 11 Min
93	1	22 Hr , 34 Min	22 Hr , 34 Min	3	0 Hr , 23 Min
94	2	22 Hr , 34 Min	22 Hr , 34 Min	1	23 Hr , 25 Min
95	4	22 Hr , 36 Min	22 Hr , 36 Min	2	23 Hr , 29 Min
96	1	22 Hr , 37 Min	22 Hr , 37 Min	3	0 Hr , 28 Min
97	2	22 Hr , 43 Min	22 Hr , 43 Min	2	0 Hr , 2 Min
98	1	22 Hr , 44 Min	22 Hr , 44 Min	1	23 Hr , 25 Min
99	2	22 Hr , 44 Min	22 Hr , 44 Min	2	23 Hr , 54 Min
100	8	22 Hr , 45 Min	22 Hr , 45 Min	2	0 Hr , 2 Min
101	8	22 Hr , 46 Min	22 Hr , 46 Min	1	23 Hr , 26 Min
102	2	22 Hr , 46 Min	22 Hr , 46 Min	2	23 Hr , 56 Min
103	2	22 Hr , 47 Min	no entry	0	22 Hr , 47 Min
104	3	22 Hr , 47 Min	no entry	0	22 Hr , 47 Min
105	4	22 Hr , 47 Min	no entry	0	22 Hr , 47 Min
106	3	22 Hr , 49 Min	no entry	0	22 Hr , 49 Min
107	3	22 Hr , 51 Min	22 Hr , 55 Min	2	0 Hr , 24 Min
108	2	22 Hr , 51 Min	no entry	0	22 Hr , 55 Min
109	2	22 Hr , 53 Min	no entry	0	22 Hr , 55 Min
110	4	22 Hr , 53 Min	no entry	0	22 Hr , 55 Min
111	6	22 Hr , 54 Min	no entry	0	22 Hr , 55 Min
112	4	22 Hr , 55 Min	no entry	0	22 Hr , 55 Min
113	2	22 Hr , 56 Min	22 Hr , 56 Min	6	1 Hr , 43 Min
114	2	22 Hr , 57 Min	no entry	0	22 Hr , 57 Min
115	1	22 Hr , 57 Min	22 Hr , 57 Min	1	23 Hr , 37 Min
116	2	23 Hr , 1 Min	no entry	0	23 Hr , 1 Min
117	2	23 Hr , 2 Min	no entry	0	23 Hr , 2 Min
118	2	23 Hr , 22 Min	23 Hr , 25 Min	4	1 Hr , 24 Min
119	2	23 Hr , 36 Min	23 Hr , 36 Min	2	0 Hr , 32 Min
120	4	23 Hr , 39 Min	23 Hr , 39 Min	1	0 Hr , 23 Min

Groupe Numero	Nombre Clients	Heure d'Arrivee	Heure d'Entree	Nombre de Tournees	Heure de Sortie
121	2	23 Hr , 44 Min	23 Hr , 44 Min	5	2 Hr , 26 Min
122	4	23 Hr , 53 Min	23 Hr , 54 Min	2	1 Hr , 0 Min
123	2	23 Hr , 54 Min	23 Hr , 54 Min	1	0 Hr , 12 Min
124	2	0 Hr , 0 Min	0 Hr , 0 Min	3	1 Hr , 14 Min
125	2	0 Hr , 3 Min	0 Hr , 3 Min	1	0 Hr , 48 Min
126	2	0 Hr , 4 Min	0 Hr , 4 Min	4	2 Hr , 4 Min
127	4	0 Hr , 9 Min	0 Hr , 9 Min	4	2 Hr , 1 Min
128	2	0 Hr , 12 Min	0 Hr , 12 Min	2	1 Hr , 9 Min
129	2	0 Hr , 18 Min	0 Hr , 23 Min	3	2 Hr , 36 Min
130	2	0 Hr , 21 Min	0 Hr , 23 Min	1	0 Hr , 41 Min
131	2	0 Hr , 22 Min	0 Hr , 23 Min	1	0 Hr , 49 Min
132	2	0 Hr , 23 Min	0 Hr , 23 Min	1	1 Hr , 9 Min
133	2	0 Hr , 25 Min	0 Hr , 25 Min	3	2 Hr , 21 Min
134	4	0 Hr , 25 Min	0 Hr , 25 Min	1	1 Hr , 5 Min
135	1	0 Hr , 37 Min	0 Hr , 37 Min	2	2 Hr , 7 Min
136	2	0 Hr , 37 Min	0 Hr , 37 Min	3	2 Hr , 29 Min
137	2	0 Hr , 44 Min	0 Hr , 48 Min	3	2 Hr , 33 Min
138	4	0 Hr , 46 Min	0 Hr , 48 Min	4	2 Hr , 25 Min
139	6	0 Hr , 50 Min	0 Hr , 50 Min	2	2 Hr , 10 Min
140	2	0 Hr , 50 Min	0 Hr , 50 Min	2	1 Hr , 44 Min
141	2	0 Hr , 51 Min	0 Hr , 51 Min	2	2 Hr , 8 Min
142	3	0 Hr , 56 Min	0 Hr , 56 Min	1	1 Hr , 35 Min
143	2	0 Hr , 57 Min	0 Hr , 57 Min	2	1 Hr , 59 Min
144	1	0 Hr , 58 Min	0 Hr , 58 Min	3	2 Hr , 24 Min

Numero du Spectacle	Numero de la Stripteaseuse	Heure de Debut	Heure de Fin
1	1	18 Hr 0 Mn	18 Hr 12 Mn
2	2	18 Hr 27 Mn	18 Hr 35 Mn
3	1	19 Hr 0 Mn	19 Hr 5 Mn
4	2	19 Hr 24 Mn	19 Hr 33 Mn
5	1	19 Hr 52 Mn	19 Hr 59 Mn
6	2	20 Hr 22 Mn	20 Hr 30 Mn
7	1	20 Hr 53 Mn	21 Hr 6 Mn
8	2	21 Hr 22 Mn	21 Hr 33 Mn
9	1	21 Hr 50 Mn	21 Hr 55 Mn
10	2	22 Hr 19 Mn	22 Hr 25 Mn
11	1	22 Hr 50 Mn	22 Hr 55 Mn
12	2	23 Hr 12 Mn	23 Hr 25 Mn
13	1	23 Hr 49 Mn	23 Hr 54 Mn
14	2	0 Hr 13 Mn	0 Hr 23 Mn

4. EXERCICE USINE-ATELIER.

4.1. ENONCE.

Une usine possède 16 machines-outils identiques dont 10 sont utilisées jusqu'au moment où elles se brisent. Il y a de la place à l'extérieur de l'usine pour les machines non utilisées.

Chaque machine travaille de façon continue jusqu'à ce qu'elle se brise. La vie moyenne d'une machine est de 250 heures. Lorsqu'une machine se brise, elle est enlevée et placée à l'extérieur avec les autres machines brisées. Ce travail prend une heure. Ensuite, on la remplace par une autre, s'il en reste encore. Le remplacement prend deux heures pour chaque machine (deux heures de démontage de la machine en panne et deux heures de montage pour la nouvelle machine).

Dès qu'il y a quatre machines ayant besoin de réparation, on demande un camion pour les transporter à l'atelier de réparation. Le chargement du camion prend une heure, ainsi que le déchargement. Le voyage entre l'atelier et l'usine dure 3 heures.

L'atelier est toujours ouvert et les machines sont remplacées dès que possible. D'abord, chaque machine doit passer par un point "A", puis un point "B", puis un point "C". Les temps de réparation aux différents points sont donnés par la figure ci-dessus.

point "A"		point "B"	point "C"
temps	probabilité	[4.0,8.0]	3.0
10.0	0.05		
11.0	0.20		
12.0	0.50		
13.0	0.20		
14.0	0.05		

Remarques:

- Il y a 3 points "A", 2 points "B" et 1 point "C".
- Chaque point s'occupe d'une machine à la fois.

- Quand les 4 machines sont réparées, on demande un camion qui les ramènera à l'usine. Les temps de transport, de chargement et de déchargement sont les mêmes que pour l'aller.

- A minuit le 31 décembre 1981, il y a 10 machines neuves en service dans l'usine et 6 de disponibles à l'extérieur.

- Il n'y a qu'un camion affecté au transport des machines réparées et à réparer. Le camion fait donc la navette entre l'usine et l'atelier. Une fois arrivé à destination, il attend une demande. Si la demande vient de l'autre endroit, il doit s'y rendre avant que le chargement puisse avoir lieu.

- L'usine et l'atelier travaillent 24 heures par jour et 365 jours par an.

- On demande de simuler pour 1 an.

4.2. PROGRAMME EN SIMUFOR.

```
      program repar
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /io/ input,output
      common /global/ u,lieuca,usine,repa,repb,repc,sort1,sort2
      logical lieuca
      external machin
c
      call init
      u = 197731723
c
      lieuca = .true.
      loc = locf(machin)
      mach = class(loc,8)
c
      usine = newst ('usine ',10)
      repa  = newst ('reparation a',3)
      repb  = newst ('reparation b',2)
      repc  = newfac ('reparation c')
      sort1 = newgr (4)
      sort2 = newgr (4)
c
      do 10 i=1,16,1
            call activ (new(mach),delay,0.0)
   10       continue
c
      call hold (8760.0)
c
c
      call storep
      call facrep
c
      stop
      end
c*************************************************************
      subroutine machin
c
      implicit integer (a-z)
      common /global/ u,lieuca,usine,repa,repb,repc,sort1,sort2
      logical lieuca
      real tv,ta,tb
      real normal
c
  100 continue
      call entst (usine,1)
      call hold (2.0)
      tv = normal (250.0,15.0,u)
      call hold (tv)
      call hold (2.0)
      call least (usine,1)
c
      call join (sort1)
      if (.not. lieuca) call hold (3.0)
      lieuca = .true.
      call hold (1.0)
      call hold (3.0)
```

```
      lieuca = .false.
      call hold (1.0)
c
      call entst (repa,1)
      tl = randin (0,100,u)
      if (tl .ge.  0 .and. tl .le.   5) ta = 10.0
      if (tl .gt.  5 .and. tl .le.  25) ta = 11.0
      if (tl .gt. 25 .and. tl .le.  75) ta = 12.0
      if (tl .gt. 75 .and. tl .le.  95) ta = 13.0
      if (tl .gt. 95 .and. tl .le. 100) ta = 14.0
      call hold (ta)
      call least (repa,1)
c
      call entst (repb,1)
      tb = float (randin(4,8,u))
      call hold (tb)
      call least (repb,1)
c
      call entfac (repc)
      call hold(3.0)
      call leafac (repc)
c
      call join (sort2)
      if (lieuca) call hold (3.0)
      lieuca = .false.
      call hold (1.0)
      call hold (3.0)
      lieuca = .true.
      call hold (1.0)
c
      go to 100
c
  999 continue
      return
      end
```

4.3. RESULTATS.

A: Stations multiples: Storages

* 1!... Informations d'ordre general !...

Identification	Nb entrees	Nb entrees sans attente	% entrees sans attente	Tps observ des statist	Tps utilis de la station	Taux utilis de la station
reparation b	340	255	75.00	8760.000	1409.698	16.09
reparation a	340	254	74.71	8760.000	1959.025	22.36
usine	352	36	10.23	8760.000	8760.000	*****

* 2!... Statistiques sur le nombre d'unites demandees par transaction !...

Identification	Nombre des observations	Demande minimale	Demande maximale	Demande moyenne	Ecart-type de la demande
reparation b	340	1	1	1.000	0.000
reparation a	340	1	1	1.000	0.000
usine	352	1	1	1.000	0.000

* 3!... Statistiques sur le temps d'attente devant le serveur

(conditionnelle au fait qu'il y ait effectivement eu une attente) !...

Identification	Nombre des observations	Attente minimale	Attente maximale	Attente moyenne	Ecart-type de l'attente
reparation b	85	0.001	7.000	4.012	1.491
reparation a	86	7.011	12.000	11.197	0.873
usine	314	0.039	258.702	84.299	51.635

* 4!... Statistiques sur la longueur de la file d'attente devant le serveur !...

Identification	Longueur initiale	Longueur minimale	Longueur maximale	Longueur moyenne	Ecart-type de longueur	Longueur courante
reparation b	0	0	1	0.039	0.193	0
reparation a	0	0	2	0.110	0.315	0
usine	0	0	6	3.037	1.750	0

* 5!... Statistiques sur le nombre d'unites occupees de la station !...

Identification	Capacite de la station	Occupation initiale	Occupation minimale	Occupation maximale	Occupation moyenne	Ecart-type d'occupation	Occupation courante
reparation b	2	0	0	2	0.231	0.544	1
reparation a	3	0	0	3	0.464	0.980	0
usine	10	0	6	10	9.959	0.261	10

* 6!... Statistiques sur le temps de service d'une transaction par le serveur !...

Identification	Nombre des observations	Service minimal	Service maximal	Service moyen	Ecart-type du service
reparation b	339	4.000	8.000	5.962	1.464
reparation a	340	10.000	14.000	11.950	0.933
usine	340	196.123	301.434	252.476	14.527

B: Stations simples: Facilities

1) Informations d'ordre general :

Identification	Nb entrees	Nb entrees sans attente	% entrees sans attente	Tps observ des statist	Tps utilis de la station	Taux utilis de la station
reparation c	339	197	58.11	8760.000	1015.713	11.59

2) Statistiques sur le temps d'attente devant le serveur (conditionnelle au fait qu'il y ait effectivement eu une attente) :

Identification	Nombre des observations	Attente minimale	Attente maximale	Attente moyenne	Ecart-type de l'attente
reparation c	142	0.000	3.000	1.472	0.983

3) Statistiques sur la longueur de la file d'attente devant le serveur :

Identification	Longueur initiale	Longueur minimale	Longueur maximale	Longueur moyenne	Ecart-type de longueur	Longueur courante
reparation c	0	0	1	0.024	0.153	0

4) Statistiques sur le temps de service d'une transaction par le serveur :

Identification	Nombre des observations	Service minimal	Service maximal	Service moyen	Ecart-type du service
reparation c	338	3.000	3.000	3.000	0.000

5. EXERCICE RESEAU.

5.1. ENONCE.

Soit un réseau d'ordinateur. Sa topologie est:

- formé en deux étoiles de 5 noeuds plus un central, ces derniers étant reliés par une ligne qui sera toujours "full-duplex";
- les autres lignes "full-duplex".

Les messages ont une longueur équidistribuée entre 1 et 133 octets. Le traffic externe est de 1 message par unité de temps et par noeud, suivant la période:

- 20 h. - 8 h. : traffic minimum, moyenne = 10 % de la moyenne générale;
- 8 h. - 20 h. : distribution normale.

La destination des messages est aléatoire.

Le temps de traitement d'un message dans un central est équidistribué entre 0.5 et 1.5 unités.

Les entrés-sorties sont faites en DMA et n'entrent pas dans le temps de traitement.

Un central peut traiter 3 messages à la fois.

L'unité de temps est la seconde.

La simulation proposée porte sur une durée de 100 minutes en service de nuit (traffic minimum).

5.2. PROGRAMME EN SIMUFOR.

```
      program reseau
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ bidon(7),num(1)
      common /io/ input,output
      common /global/ germ1,germ2,germ3,germ4,germ5,sn,a,b,moy
      common /global/ vittrm,vittlc,tsim,sortie,i,j,n,his
      common /global/ ndperi,messag
      common /entite/ ligtrm(10,2),ligct(2),ndct(2)
      common /locall/ locl,loc2,noeud
c
      logical sn
      real a,b,moy,tsim
      external msg,ndgm
c
      call init
c
      sn = .true.
      vittrm = 1200
      vittlc = 3000
      n = 3
      a = 0.5
      b = 1.5
      moy = 1.0
      tsim = 100.0
      germ1 = 5412893
      germ2 = 2156035
      germ3 = 2021597
      germ4 = 2456011
      germ5 = 2105909
c
      locl = locf (msg)
      messag = class(locl,16)
      loc2 = locf (ndgm)
      ndperi = class(loc2,9)
c
      his = newhis ('temps entre mes',10,0.0,(10.0*moy))
      do 100 i=1,2
           ligct(i) = newfac('ligne centrale')
  100 continue
      do 200 i=1,10
           ligtrm(i,1) = newfac('ligne peri in')
           ligtrm(i,2) = newfac('ligne peri out')
  200 continue
      do 300 i=1,2
           ndct(i) = newst('noeud central',n)
  300 continue
c
      do 400 i=1,10
           noeud = new(ndperi)
           num(noeud) = i
           call activ(noeud,delay,0.0)
  400 continue
c
      call hold(60.0*tsim)
c
```

```
      call hisrep
      call facrep
      call storep
c
  999 continue
      stop
      end
c
c
c
c
c

      subroutine msg
c
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ bidon(7),orig(1),dest(1),long(1),ndorig(1)
      common /objatr/ nddest(1),tpstrm(1),tpstlc(1),tpstrt(1)
      common /objatr/ sens(1)
      common /io/ input,output
      common /global/ germ1,germ2,germ3,germ4,germ5,sn,a,b,moy
      common /global/ vittrm,vittlc,tsim,sortie,i,j,n,his
      common /global/ ndperi,messag
      common /entite/ ligtrm(10,2),ligct(2),ndct(2)
      common /local2/ aleat
c
      logical sn
      real a,b,moy,tsim
      real tpstrm,tpslc,tpstrt
      real unif
c
      long(actuel) = randin(1,133,germ1)
      tpstrm(actuel) = float(long(actuel)*8)/float(vittrm)
      aleat = randin(1,9,germ2)
      if (aleat.lt.orig(actuel)) go to 1100
           dest(actuel) = aleat + 1
           go to 1200
 1100      continue
           dest(actuel) = aleat
 1200      continue
c
      call entfac(ligtrm(orig(actuel),1))
      call hold(tpstrm(actuel))
      call leafac(ligtrm(orig(actuel),1))
c
      ndorig(actuel) = ifix((float(orig(actuel)-1)/5.0)+1.0)
      call entst(ndct(ndorig(actuel)),1)
      tpstrt(actuel) = unif(a,b,germ3)
      call hold(tpstrt(actuel))
      call least(ndct(ndorig(actuel)),1)
c
      nddest(actuel) = ifix((float(dest(actuel)-1)/5.0)+1.0)
      if (nddest(actuel).eq.ndorig(actuel)) go to 2100
c
           if (ndorig(actuel).eq.1) go to 2110
                sens(actuel) = 2
                go to 2120
 2110           continue
```

```
                sens(actuel) = 1
 2120           continue
          call entfac(ligct(sens(actuel)))
          tpstlc(actuel) = float(long(actuel)*8)/float(vittlc)
          call hold(tpstlc(actuel))
          call leafac(ligct(sens(actuel)))
c
          call entst(ndct(nddest(actuel)),1)
          tpstrt(actuel) = unif(a,b,germ4)
          call hold(tpstrt(actuel))
          call least(ndct(nddest(actuel)),1)
c
          call entfac(ligtrm(dest(actuel),2))
          call hold(tpstrm(actuel))
          call leafac(ligtrm(dest(actuel),2))
          go to 2200
c
 2100     continue
          call entfac(ligtrm(dest(actuel),2))
          call hold(tpstrm(actuel))
          call leafac(ligtrm(dest(actuel),2))
c
 2200     continue
c
      call endpro
c
  999 continue
      return
      end
c
c
c
c
c
      subroutine ndgm
      implicit integer (a-z)
      common /sysvar/ none,head,actuel,after,before,delay,at,main
      common /objatr/ bidon(7),num(1),intmes(1)
      common /io/ input,output
      common /global/ germ1,germ2,germ3,germ4,germ5,sn,a,b,moy
      common /global/ vittrm,vittlc,tsim,sortie,i,j,n,his
      common /global/ ndperi,messag
      common /entite/ ligtrm(10,2),ligct(2),ndct(2)
      common /local3/ mes
c
      logical sn
      real a,b,moy,tsim
      real intmes
      real negexp
      dimension orig(1)
      equivalence (orig,num)
c
   10 continue
c
      if (sn.eq..true.) go to 20
           intmes(actuel) = negexp(1.0/moy,germ5)
           go to 30
   20      continue
           intmes(actuel) = negexp(1.0/(10.0*moy),germ5)
```

```
   30      continue
      call addhis(his,intmes(actuel))
c
      call hold(intmes(actuel))
c
      mes = new(messag)
      orig(mes) = num(actuel)
      call activ(mes,delay,0.0)
c
      go to 10
c
  999 continue
      return
      end
```

5.3. RESULTATS.

A: Stations multiples: Storages

1: Informations d'ordre general :

Identification	Nb entrees	Nb entrees sans attente	% entrees sans attente	Tps observ des statist	Tps utilis de la station	Taux utilis de la station
noeud central	4719	4479	94.91	6000.000	3272.110	54.54
noeud central	4694	4498	95.82	6000.000	3324.689	55.41

2: Statistiques sur le nombre d'unites demandees par transaction :

Identification	Nombre des observations	Demande minimale	Demande maximale	Demande moyenne	Ecart-type de la demande
noeud central	4719	1	1	1.000	0.000
noeud central	4694	1	1	1.000	0.000

3: Statistiques sur le temps d'attente devant le serveur

(conditionnelle au fait qu'il y ait effectivement eu une attente) :

Identification	Nombre des observations	Attente minimale	Attente maximale	Attente moyenne	Ecart-type de l'attente
noeud central	240	0.005	1.082	0.305	0.233
noeud central	196	0.001	1.185	0.297	0.213

4: Statistiques sur la longueur de la file d'attente devant le serveur :

Identification	Longueur initiale	Longueur minimale	Longueur maximale	Longueur moyenne	Ecart-type de longueur	Longueur courante
noeud central	0	0	4	0.012	0.129	0
noeud central	0	0	4	0.010	0.114	0

5: Statistiques sur le nombre d'unites occupees de la station :

Identification	Capacite de la station	Occupation initiale	Occupation minimale	Occupation maximale	Occupation moyenne	Ecart-type d'occupation	Occupation courante
noeud central	3	0	0	3	0.793	0.871	0
noeud central	3	0	0	3	0.788	0.853	1

6: Statistiques sur le temps de service d'une transaction par le serveur :

Identification	Nombre des observations	Service minimal	Service maximal	Service moyen	Ecart-type du service
noeud central	4719	0.500	1.500	1.008	0.289
noeud central	4693	0.500	1.500	1.008	0.288

B: Stations simples: Facilities

1) Informations d'ordre general

Identification	Nb entrees	Nb entrees sans attente	% entrees sans attente	Tps observ des statist	Tps utilis de la station	Taux utilis de la station
ligne peri out	589	565	95.93	6000.000	269.913	4.50
ligne peri in	583	557	95.54	6000.000	256.700	4.28
ligne peri out	621	590	95.01	6000.000	276.413	4.61
ligne peri in	622	588	94.53	6000.000	273.826	4.56
ligne peri out	572	547	95.63	6000.000	251.786	4.20
ligne peri in	575	556	96.70	6000.000	250.013	4.17
ligne peri out	663	621	93.67	6000.000	294.582	4.91
ligne peri in	594	565	95.12	6000.000	263.253	4.39
ligne peri out	588	561	95.41	6000.000	256.467	4.27
ligne peri in	615	588	95.61	6000.000	278.693	4.64
ligne peri out	572	545	95.28	6000.000	253.733	4.23
ligne peri in	586	541	92.32	6000.000	263.909	4.40
ligne peri out	598	572	95.65	6000.000	271.387	4.52
ligne peri in	591	568	95.78	6000.000	267.167	4.45
ligne peri out	591	566	95.77	6000.000	261.285	4.35
ligne peri in	616	586	95.13	6000.000	287.020	4.78
ligne peri out	625	601	96.16	6000.000	282.213	4.70
ligne peri in	615	588	95.61	6000.000	276.066	4.60
ligne peri out	577	555	96.19	6000.000	261.513	4.35
ligne peri in	599	577	96.33	6000.000	264.100	4.40
ligne centrale	1686	1686	100.00	6000.000	0.000	0.00
ligne centrale	1730	1730	100.00	6000.000	0.000	0.00

2) Statistiques sur le temps d'attente devant le serveur

(conditionnelle au fait qu'il y ait effectivement eu une attente)

Identification	Nombre des observations	Attente minimale	Attente maximale	Attente moyenne	Ecart-type de l'attente
ligne peri out	24	0.013	0.804	0.335	0.231
ligne peri in	26	0.047	0.849	0.342	0.201
ligne peri out	31	0.018	1.141	0.394	0.280
ligne peri in	34	0.020	0.732	0.320	0.223
ligne peri out	25	0.009	0.635	0.283	0.194
ligne peri in	19	0.002	0.670	0.303	0.192
ligne peri out	42	0.008	0.799	0.361	0.226
ligne peri in	29	0.014	1.006	0.298	0.221
ligne peri out	27	0.011	0.753	0.285	0.205
ligne peri in	27	0.000	0.705	0.264	0.203
ligne peri out	27	0.022	1.017	0.352	0.278
ligne peri in	45	0.018	0.905	0.296	0.241
ligne peri out	26	0.018	0.878	0.323	0.244
ligne peri in	25	0.012	0.724	0.325	0.176
ligne peri out	25	0.020	0.844	0.293	0.190
ligne peri in	30	0.006	1.382	0.260	0.276
ligne peri out	24	0.008	0.801	0.321	0.232
ligne peri in	27	0.013	0.751	0.297	0.209
ligne peri out	22	0.014	0.786	0.289	0.232
ligne peri in	22	0.022	1.011	0.315	0.259
ligne centrale	0	*********	0.000	0.000	0.000
ligne centrale	0	*********	0.000	0.000	0.000

3: Statistiques sur la longueur de la file d'attente devant le serveur :

Identification	Longueur initiale	Longueur minimale	Longueur maximale	Longueur moyenne	Ecart-type de longueur	Longueur courante
ligne peri out	0	0	1	0.001	0.037	0
ligne peri in	0	0	1	0.001	0.038	0
ligne peri out	0	0	2	0.002	0.048	0
ligne peri in	0	0	2	0.002	0.044	0
ligne peri out	0	0	1	0.001	0.034	0
ligne peri in	0	0	1	0.001	0.031	0
ligne peri out	0	0	2	0.003	0.051	0
ligne peri in	0	0	2	0.001	0.040	0
ligne peri out	0	0	1	0.001	0.036	0
ligne peri in	0	0	2	0.001	0.034	0
ligne peri out	0	0	2	0.002	0.042	0
ligne peri in	0	0	2	0.002	0.049	0
ligne peri out	0	0	2	0.001	0.041	0
ligne peri in	0	0	1	0.001	0.037	0
ligne peri out	0	0	2	0.001	0.036	0
ligne peri in	0	0	2	0.001	0.038	0
ligne peri out	0	0	1	0.001	0.036	0
ligne peri in	0	0	2	0.001	0.038	0
ligne peri out	0	0	2	0.001	0.033	0
ligne peri in	0	0	2	0.001	0.036	0
ligne centrale	0	0	0	0.000	0.000	0
ligne centrale	0	0	0	0.000	0.000	0

4: Statistiques sur le temps de service d'une transaction par le serveur :

Identification	Nombre des observations	Service minimal	Service maximal	Service moyen	Ecart-type du service
ligne peri out	589	0.007	0.887	0.458	0.259
ligne peri in	583	0.007	0.887	0.440	0.254
ligne peri out	621	0.007	0.887	0.445	0.252
ligne peri in	622	0.007	0.887	0.440	0.249
ligne peri out	572	0.007	0.887	0.440	0.260
ligne peri in	575	0.007	0.887	0.435	0.258
ligne peri out	662	0.007	0.887	0.445	0.258
ligne peri in	594	0.007	0.887	0.443	0.256
ligne peri out	588	0.007	0.887	0.436	0.257
ligne peri in	615	0.007	0.887	0.453	0.254
ligne peri out	572	0.007	0.887	0.444	0.251
ligne peri in	585	0.007	0.887	0.451	0.259
ligne peri out	508	0.007	0.887	0.454	0.239
ligne peri in	583	0.007	0.887	0.451	0.258
ligne peri out	590	0.007	0.887	0.442	0.256
ligne peri in	616	0.007	0.887	0.466	0.247
ligne peri out	625	0.007	0.887	0.452	0.256
ligne peri in	615	0.007	0.887	0.449	0.257
ligne peri out	577	0.007	0.887	0.453	0.256
ligne peri in	599	0.007	0.887	0.441	0.254
ligne centrale	1686	0.000	0.000	0.000	0.000
ligne centrale	1730	0.000	0.000	0.000	0.000

D: Histogrammes

Histogramme : temps entre mes.

Intervalle num : 1	[- infini , 0.000 [	nombre d'occurences :	0
Intervalle num : 2	[0.000 , 10.000 [	nombre d'occurences :	3808
Intervalle num : 3	[10.000 , 20.000 [	nombre d'occurences :	1374
Intervalle num : 4	[20.000 , 30.000 [	nombre d'occurences :	503
Intervalle num : 5	[30.000 , 40.000 [	nombre d'occurences :	211
Intervalle num : 6	[40.000 , 50.000 [	nombre d'occurences :	71
Intervalle num : 7	[50.000 , 60.000 [	nombre d'occurences :	23
Intervalle num : 8	[60.000 , 70.000 [	nombre d'occurences :	15
Intervalle num : 9	[70.000 , 80.000 [	nombre d'occurences :	1
Intervalle num : 10	[80.000 , 90.000 [	nombre d'occurences :	1
Intervalle num : 11	[90.000 , 100.000 [	nombre d'occurences :	1
Intervalle num : 12	[100.000 , + infini [	nombre d'occurences :	0